conference

proceedings

Large Installation System Administration of Windows NT Conference Proceedings

Seattle, Washington
August 5–8, 1998

Co-sponsored by **The USENIX Association and SAGE, the System Administrators Guild**

USENIX®
The Advanced Computing Systems Association

SAGE
THE SYSTEM ADMINISTRATORS GUILD

Large Installation System Administration of Windows NT Conference
Seattle, Washington August 1998

For additional copies of these proceedings contact:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Phone: 510 528 8649
FAX: 510 548 5738
Email: office@usenix.org
WWW URL: http://www.usenix.org

The price is $18 for members and $24 for nonmembers.
Outside the U.S.A. and Canada, please add
$12 per copy for postage (via air printed matter).

1998 © Copyright by The USENIX Association
All Rights Reserved

This volume is published as a collective work. Rights to individual papers remain with the author or the author's employer. Permission is granted for the noncommercial reproduction of the complete work for educational or research purposes. USENIX acknowledges all trademarks herein.

ISBN 1-880446-96-0

Printed in the United States of America on 50% recycled paper, 10-15% post consumer waste.

USENIX Association

Proceedings of the Large Installation System Administration of Windows NT Conference

August 5-8, 1998
Seattle, Washington

Conference Organizers

Program Co-Chairs
Rémy Evard, *Argonne National Laboratory*
Ian Reddy, *Cisco Systems, Inc*

Program Committee
Michael Carpenter, *Pencom Systems Administration*
Richard Holland, *Rockwell Collins, Inc.*
Kendall Martin, *Microsoft Corporation*
Chris Rouland, *Internet Security Systems, Inc.*
Phil Scarr, *Global Networking and Computing, Inc.*
Mark Verber, *WebTV Networks Futures*

The USENIX Association Staff

Table of Contents

Large Installation System Administration of Windows NT Conference

August 5-8, 1998
Seattle, Washington

Thursday, August 6

Friday, August 7

Saturday, August 8

Message from the Program Chairs

Welcome to Seattle and the Conference on Large Installation System Administration of Windows NT!

This conference is taking place amidst an industry-wide flurry of activity centered around Windows NT. Many changes have taken place in the computing world since our first workshop on Large-Scale System Administration of Windows NT, and with the release of NT 5.0 right around the corner, a great deal more is likely to change during the coming year. Yet one thing remains the same. No matter what your institution does, no matter what operating systems you run, it is still a challenge to administer a large computing environment.

These proceedings represent the work of twenty-six authors on ten different topics. They each present a different perspective on one or more of the issues that we all face. It is our hope that these papers will not only help you to solve some of your current problems, but that they will also build a foundation and example for future work, thereby helping to form a richer set of tools and techniques for the systems administration community.

We would like to acknowledge the members of our program committee. They helped to guide the direction of the conference, spent many hours soliciting and reviewing the technical papers, and organized the invited talks. In a field of systems administration that is as new as this, all of these can be formidable tasks. Also, Todd Needham and Kendall Martin of Microsoft were extraordinarily helpful in locating the right people within Microsoft to address issues raised by our community.

Finally, we would especially like to thank Ellie Young and all of the USENIX staff for their hard work organizing the conference, the proceedings, and everything else that is involved in a successful endeavor. As participants in previous USENIX events, we understood that they were important, but as co-chairs, we've learned that they have the hardest work, and do an incredible job. Thanks.

Rémy Evard and Ian Reddy, Conference Co-Chairs

Patch32 : A System for Automated Client OS Updates

Gerald Carter
Engineering Network Services
Auburn University
jerry@eng.auburn.edu

Abstract

The adage "a chain is only as strong as its weakest link" is true for network security, the link being the host on the network. To secure a network, hosts must be thoughtfully installed and kept updated with the appropriate patches. For hosts running Microsoft Windows 95® or Microsoft Windows NT Workstation® keeping patches current is problematic.

Unlike most Unix variants, neither Windows 95 nor NT Workstation ship with a network extensible update mechanism. Though third party solutions are available, they can be costly to implement for large networks. This paper presents a free update mechanism for hosts running Windows 9x or NT Workstation served by Samba (see Appendix A).

Developed to patch Microsoft's 32 bit operating systems, the name Patch32 was adopted. Patch32 was developed for an existing network dominated by Sun Microsystems' SPARC servers running Solaris®, however, Patch32 can be used in any environment that provides SMB file services.

1. Introduction

The initial design criteria for Patch32 was simple: Provide for completely automated, remotely administered updates to Microsoft's 32 bit operating systems. Essential to any automated update mechanism is the ability to determine what operating system the machine is running and what updates have already been installed so that the necessary updates can be determined. To be effective, any update mechanism must be able to access the necessary updates from a central location without human intervention and with a mechanism to guarantee that the program will be run on the client systems with reasonable regularity.

This paper addresses issues that arose during the implementations of *Patch32*. The remaining sections are organized as follows. Section 2 discusses solutions to the various requirements of such an update system. Section 3 describes security concerns related to *Patch32* and ways to address these concerns. For those wishing to implement the system, suggestions for customization are described in Section 4. Finally, lessons learned from the implementation process as well as plans for future enhancement are included in Section 5. The appendix contains the perl source code for the Patch32 script, example patch listings, URL's and other references for the reader interested in finding out more about the software and ideas presented here

2. Implementation

2.1. File Access

The file services needed to support Patch32 are provided by a dedicated 196MB Sun Ultra 170 running Solaris 2.5.1. Samba, which is available under the GNU General Public License, provides file and print services to Server Message Block (SMB) clients including Microsoft's Windows for Workgroups®, Windows 95 and Windows NT. Samba also includes the capability to act as a Primary Domain Controller (PDC) for Windows NT Domains, although this support is currently in the testing stages and is not included in the main source distribution as of version 1.9.18p7.

SMB services were chosen rather than other file sharing protocols such as NFS due to the native support for SMB within the Microsoft operating systems. This support, combined with the Microsoft Network Client, allowed built-in tools such as domain login scripts and System Policies to be used to configure the execution of patch scripts on remote machines.

Two separate copies of Samba were configured on the physical server, each have its own network interface. This was accomplished by utilizing two settings in the samba configuration file (smb.conf). The "interfaces" parameter specifies the correct network interface to which the samba processes should attach and the value

for the "socket address" parameter determines the IP address to which the samba processes should bind. The reader should be aware that using this second parameter may cause problems with network browsing. The problem has been reproduced when the domain controller is bound to the second network interface which has been created as a virtual interface. Therefore it may be necessary to use a central WINS server in order for a client to resolve names correctly even if it is on the same subnet as the samba server.

At the College of Engineering, the first samba server, \\USERSERVER, provided access to department shares, user shares, and various network printers. The second copy of samba, \\GUESTSERVER, provided access to various system and development tools such as the Java SDK, Perl5 for Win32, administrative scripts and command line utilities.

Both servers are configured with user level security however, \\GUESTSERVER is configured to only allow guest connections. This is accomplished by defining GUEST_SESSETUP to be the integer 2 in the file local.h, which is part of the samba source code distribution, and by isolating \\GUESTSERVER from the normal list of accounts. Therefore any user attempting to connect is validated as the guest account specified in the smb.conf file. This mechanism allows machines to access specific shares without validation. Since all files accessible on \\GUESTSERVER have been deemed as public access, the setup does not create a security concern.

At this point is may be obvious why it was decided to run two separate samba servers. Guest connections are not often desired to shares containing data or to printers where page accounting has been enabled. It is possible to recreate this same type of behavior without maintaining two servers, but separation was deemed necessary for management purposes.

2.2. Patch Preparation

The updates discussed in this paper were released by Microsoft and downloaded from the their web site (see Appendix A). This section will discuss the preparation of the archived files necessary to integrate the patches into the model presented here.

A system update contains three basic components. The first is the collection of updated files. This may include dynamic link libraries, executables, information files, device drivers or any other imaginable portion of the local system.

The second component is the patch installation program. Windows 95 updates normally rely on the rundll.exe and setupx.dll files, which are by default located in the \windows and \windows\system directories respectively. Windows NT Service Packs and Hotfixes normally include an installation executable.

The final necessary component for an installation program is a listing of directory locations and registry keys where the updated files and information is to be placed. This listing may be internal to the installation executable or, as is the case with both Windows 95 and Windows NT fixes, external in the form of an information file (INF).

Both the Windows 95 and the Windows NT updates released by Microsoft are packed in a self-installing executable file. The method for extracting the internal files is different for each operating system.

To install a Windows 95 update such as the Service Pack 1, a user would normally simply launch the update executable. At this point the setup program will extract the archived files into the %TEMP% directory. When prompted to continue with the update, the user may then make a copy of the files that have been extracted and cancel the setup program.

In order to extract the archived files from either a Windows NT Service Pack or Hotfix, one must simply run the archived executable from a console window and pass the appropriate parameter to the program. For example, to extract the files contained in the update named "hotfixl_i.exe", the standard method would be run "hotfixl_i.exe /x" from a command prompt. Please consult the service pack or hotfix documentation for the correct syntax.

2.3. Script Implementation

Once machines are able to the access the updated files from a central location, the next step is to automate the process of applying the patches. For this process, Perl (see Appendix A) was chosen as the implementation language. Reasons for selecting Perl include fast development time due to familiarity with the language on a Unix platform as well as the availability of modules to interface with specific Windows 95 / NT entities such as the event log and the system registry.

```
determine what OS we are running;
set the patch directory;
set the install method;
open $patchdir\$patchFile;
foreach ( entry in patch listing ) {
   split the entry into parts;
   if ( registry key does not exist ) {
      install the patch;
   }
}

if ( any patches have been installed ) {
   execute rebootMessage;
}
```

Figure 1: Patch32 algorithm

As stated previously, the first thing that is necessary to install updates to an operating system is to determine what version of the OS is currently running. The following segment of code will return all the information needed.

```
($osString,$majorVer,$minorVer,$osBuild,
$osId) = Win32::GetOSVersion ();

$osString   OS revision string
$majorVer   Major version number
$minorVer   Minor version number
$osBuild    OS build number
$osType     integer (win32s = 0; Win95
                     = 1; WinNT = 2)
```

Figure 2: Win32::GetOSVersion() and return values

The OS specific patch location is determined by testing $osType and then creating a directory path by catenating the Patch32 root directory specified at the beginning of the script with the OS name (either win95 or winnt) and $osBuild. For example, Windows 95 OSR1 has a build number of 67109814. The full path to the correct patches would be

```
$basedir\win95\67109814
```

From this patch directory, the program will open a list of updates to be applied to the client. The format of a patch listing entry is given below.

```
<registry key>:<install patch directory>
```

Figure 3: Patch entry format

Each entry is delimited by a carriage return. Lines beginning with a semicolon (;) or a pound sign (#) are ignored as comments as are blank lines. The "registry key" field is the absolute path to a key in the local system registry whose existence implies a previous application of the patch. The hive key HKEY_LOCAL_MACHINE is abbreviated as HKLM.

```
; This fix corrects GETADMIN problem
HKLM\SOFTWARE\...\Hotfix\Q146965:admnfix
```

Figure 4: Example patch listing entry. A portion of the registry key field has been deleted to prevent line wrapping.

Once the patch entry is read, it is split into its component parts. Using the given registry key, the program checks for its existence in the local registry. The nonexistence of the registry key indicates that the patch has not been installed. If it does exist, then the patch has previously been applied and the program does not attempt to install it. Because of the frequency with which the patch program will be run, it is necessary that the script be efficient when evaluating the current state of the client. Therefore no attempt is made to verify the integrity of previously installed updates or to examine the version of installed files. The "install patch directory" field is the name of the directory containing the update. This is a path relative to the previously determined OS specific patch directory and will be used during the patch installation process described in the next section.

2.4. Patch Installation

During the development of Patch32, one goal was to provide a method of deploying new patches with minimal effort. By default, all Windows NT hotfixes released by Microsoft store a patch identification number in the following registry key

```
HKEY_LOCAL_MACHINE
   Software
      Microsoft
         Windows NT
            CurrentVersion
               Hotfixes
                  <patch number>
```

Therefore it is possible to determine the presence of a hotfix by the existence of its ID in the registry. The Service Packs create a string value in

```
HKEY_LOCAL_MACHINE
   Software
```

```
        Microsoft
            Windows NT
                CurrentVersion
"CSD Version" = "<Service Pack Name>"
```

Checking for the existence of this value will only determine if any service pack has been installed rather than a specific one. Therefore the setup INF file for the Service Pack is modified to create a key in

```
HKEY_LOCAL_MACHINE
    Software
        Microsoft
            Windows NT
                CurrentVersion
                    Hotfixes
                        <SvcPackNo>
```

The existence of an installed service pack may then be determined by examining only one registry key. The same method may be used when an update creates many keys but only one need be examined.

By default, Windows 95 update information is stored in

```
HKEY_LOCAL_MACHINE
    Software
        Microsoft
            Windows
                CurrentVersion
                    Setup
                        Updates
                            UPD<patch number>
```

These default keys are used only to reduce the amount of effort needed to integrate a new patch into the existing setup. Site specific keys may be used as long as they are created by the update once it has been applied to the local system.

Once the registry key for the patch and the location from which to install the patch is known, the setup program for the update can be executed using the system() function. Patch32 allows for a single install method to be specified for Windows 95 and one for Windows NT.

In the case of Windows 95, the INF file provided with the update is used for the installation. This file is parsed using a modified version of the Windows 95 default method that is stored in the system registry.

```
rundll.exe setupx.dll
              InstallHinfSection 132
              DefaultInstall <filename>
```

Figure 5: Default Windows 95 install method for INF files.

Setupx.dll is instructed to process the "DefaultInstall" section of the file given by the last parameter. The only difference in the install method used by the Patch32 script is that the fourth parameter is 133 rather than 132. This has the effect of not prompting the user to reboot after the INF file is processed.

For Windows NT service packs and hotfixes, the update.exe utility provided with Service Pack 3 for Windows NT 4.0 can be used to install both

```
update.exe -z -u -q <filename>
```

Figure 6: Windows NT install procedure for a non-interactive update without a reboot after patch completion.

These methods are used to reduce the work needed for new patches. They are not the only possible installation methods. Custom setup programs may be may be used as long as the method to install each update per OS is the same. Modifications to the patch installation method are discussed in Section 4.

The Patch32 script continues until all entries have been processed. At the end of the program, if any patches have been installed, the contents of a string variable which specify a reboot message are executed by another call to the system() function. Windows 95 clients are informed that the system has been patched and changes will take effect after the next system reboot. Windows NT displays a message that the system will reboot due to patches having been applied and is restarted using the shutdown utility included in the *Windows NT 4.0 Server Resource Kit*[1].

(a) `$basePatchDir\win95\w95upd.exe`

(b)
```
start $basePatchDir\winnt\shutdown
    /l /y /r /t:30 "Patches completed.
    System rebooting."
```

Figure 7: **(a)** Windows 95 and **(b)** Windows NT reboot messages executed if any updates were applied.

2.5. Script Execution

The last issue to be resolved is to guarantee that the client machines will execute the patch script on a regular basis. This program must be run during the operating system boot process but after network support has been enabled.

Windows 95 clients are able to run the program during a domain login script or from the

```
HKEY_LOCAL_MACHINE
    Software
        Microsoft
            Windows
                RunOnce
```

registry key which can be set using a remote update of the system policies.

In order to install updates on Windows NT clients, the update program must have administrative privileges on the local host. Therefore the patch program may not be run during a normal user's domain login script. The *Windows NT 4.0 Server Resource Kit* contains a service utility named AutoexNT that, upon start up, executes a batch file named %SYSTEMROOT%\System32\AutoexNT.bat. By configuring the service to start up automatically, AutoexNT.bat will be executed during the boot process. It is from this batch file that the Patch32 script is launched. By running the AutoexNT service under a local administrative account, the program is able to update system files and settings.

```
@echo off
set LOG=%SYSTEMROOT%\log\autoexnt.log
set PERL=\\GUESTSERVER\perl\bin\perl.exe
set PATCH32=\\GUESTSERVER\bin\patch32.pl


echo :::::::::::::::::::: >> %LOG%
date /t >> %LOGFILE%
time /t >> %LOGFILE%
echo :::::::::::::::::::: >> %LOG%
net start LanManWorkstation >> %LOG%
%PERL%  %PATCH32% >> %LOG%
```

Figure 8: Example AutoexNT.bat script

3. Security

3.1. Permissions

When run under Windows NT, the patch program must run as a local administrative account in order to update system files. This results in three security issues that must be addressed. These are the integrity of the patch files themselves and their associated setup programs, the security of the registry keys which are defined by the system updates and the security of the batch file run by the AutoexNT service. Obviously the last two issues are irrelevant in Windows 95 since such access control does not exist and because the AutoexNT service is not used.

In order to insure the integrity of the update packages, write access to the files should be restricted to administrative accounts only. Without this protection, a virus or Trojan Horse program could easily be distributed to clients by modifying the patch files on the server.

It is also important to maintain the security of the system registry keys created by the updates because these are the only means by which the Patch32 program can determine whether or not an update has been installed. If the security of these keys were comprised, it would be possible to avoid the installation of an update simply by creating the correct registry key. It is important to remember that the patch script makes no attempt to verify the status of currently installed patches. It is not technically difficult to perform this type of system examination, but would result in an increased execution time and degraded system performance.

The last security issue to address is the AutoexNT service. As stated previously, this service runs as a local administrative account and executes the contents of the AutoexNT.bat file whatever they may be. It is therefore extremely important that only administrative accounts be allowed to modify this file for reasons similar to those given for ensuring the integrity of updates packages.

3.2. Logging

In a distributed environment, it is important not only to be able deploy updates effectively, but also to be able to determine which machines were successfully updated and those which failed. Patch32 supports log-

ging to the Windows NT EventLog via Perl's Win32::WriteEventLog() function.

3000	The patch32 script completed successfully
3001	The patch32 script failed
3002	The %1 patch was installed successfully
3003	The %1 patch installation failed
3004	The system was rebooted due to patches having been applied

Figure 9: Patch32 EventID's and descriptions.

4. Customization

During the initial installation, Patch32 requires that certain variables, which determine paths to OS updates, be defined. These variables, located at the beginning of the source file, allow the administrator to specify the base location of the Patch32 directory hierarchy and the install methods and parameters for Windows 95 and Windows NT updates. The server administrator may also choose to modify the reboot message that is executed upon the successful application of an update.

One advantage to Patch32, besides the fact the all software components are freely available, is that the patch script itself is very short, about two hundred lines at the time of this writing. Because the code is easy to understand, site specific changes can be made easily. This makes Patch32 extremely flexible. For example, one of the many, freely available SMTP console mail programs such as Blat (see Appendix A) could be used to augment the system's current logging abilities. In this way, network administration would be notified via e-mail upon update completions or failures.

The patch installation method is also very flexible. The current configuration at the College of Engineering uses methods available with the updates released by Microsoft. A simple alternative to this would be to use a batch script named "update.bat" that would execute specific patch installation programs. This would provide the ability to have an individual installation mechanism for each OS patch. The only restrictions would be that the update program runs non-interactively under Windows NT and that it creates the registry key necessary to allow future executions of Patch32 to determine its existence.

Another possibility of expanding the current installation method would be to determine it based on the OS build number. Then each version for Windows 95 and Windows NT would have a separate method for installing updates. This would be most helpful when supporting Windows NT 3.51 and 4.0 clients from the same server.

5. Conclusion and Future Development

In conclusion, the Patch32 system provides a flexible means of deploying system updates to Windows 95 and Windows NT clients on a regular basis. Thus far Patch32 has been implemented to manage Windows NT 4.0 and Windows 95 clients, but has also been tested with Windows NT 3.51, Windows NT 5.0 Beta 1 and Windows 98 clients. The current system supports over two hundred Windows 95 machines and approximately forty Windows NT 4.0 Workstations.

One issue that arose from the experience of implementing this system is the question of 'What is a patch?' A system update is really nothing more that a reconfiguration of the client machine. Files and registry keys are created, updated and deleted. In essence, this is software installation. It is perceivable that the Patch32 system could be used to configure software packages on client machines. This, however, is left as an exercise for the reader.

Appendix A: URL's

This section lists relevant URLs for freely available software discussed in this paper.

Samba is available under the GNU General Public License and comes with full source code. Information is available at http://samba.anu.edu.au/samba.

The *Microsoft OS updates* for Windows 95 and Windows NT are available from Microsoft's web server and FTP server. See http://support.microsoft.com and ftp://ftp.microsoft.com/bussys/winnt/winnt-public/fixes.

ActiveState distributes *Perl5 for Win32* which available at http://www.ActiveState.com.

All source code for the *Patch32* system are available on-line as will be future versions. See

http://www.eng.auburn.edu/users/cartegw/Patch32 for information about obtaining the files.

Blat is a public domain console SMTP mailer for the win32 platform. It is designed to run under Windows NT and comes with full source code. Visit http://gepasi.dbs.aber.ac.uk/softw/blat.html for more information.

References

[1] Windows NT 4.0 Server Resource Kit, Microsoft Press, Redmond, Washington, 1996. ISBN 1-57231-344-7

Appendix B: Patch32 source listing

```
########################################################################
#       Filename            : patch32.pl
#       Author              : Gerald ( Jerry ) Carter
#                             jerry@eng.auburn.edu
#       Date Created        : November 17, 1997
#       Last Update         : June 6, 1998
#
#       ATTN: Some lines have been wrapped for readability.  The real
#       source may be downloaded from
#            http://www.eng.auburn.edu/users/cartegw/Patch32
#
#       Perl5 for Win32 script to patch Windows 95 / NT clients on the
#       College of Engineering network.  Client supported are
#            Windows 95 OSR1, OSR2, OSR2.1
#            Windows 98
#            Windows NT Workstation 3.51, 4.0, 5.0 Beta 1
#
#       The following variables must be set for each OS
#            $basePatchDir            Base directory of all patches
#            $patchDirectory          Location of the patch files
#            $installPatch            Patch installation method
#            $installParameter        Patch install parameters
#            $rebootMessage           win32 command that will run if any
#                                        patches are installed
#
#       $installPatch . $patchDirectory . '\' . $pSourceDir .
#           '\' . $installPatchParameter
#
#       ================================================================
#       LICENSE
#       ================================================================
#       This program is free software; you can redistribute it
#       and/or modify it under the terms of the GNU General Public
#       License as published by the Free Software Foundation; either
#       version 2 of the License, or (at your option) any later version.
#
#       This program is distributed in the hope that it will be useful,
#       but WITHOUT ANY WARRANTY; without even the implied warranty of
#       MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#       GNU General Public License for more details.
#
#       You should have received a copy of the GNU General Public
#       License along with this program; if not, write to the Free
#       Software Foundation, Inc., 675 Mass Ave, Cambridge,
#       MA 02139, USA.
#
########################################################################


# Needed module for registry functions
use Win32::Registry;
```

```
# Setup some script variables
$DEBUG = 1;
$reboot = 0;
$eventLog = 0;
$eventSource = "Patch32";
$computer = $ENV{'COMPUTERNAME'};
$eventLogFile = "Application";

########################################################################
# Script variables - Fill these in with values appropriate for
#                    your site

# Base patch directory
$basePatchDir = "\\\\ivy\\patch32";

# Win95 patch install method
$win95InstallPatch = 'rundll.exe setupx.dll,InstallHinfSection
                      DefaultInstall 133 ';

# Win95 patch install method parameter
$win95InstallPatchParm = 'install.inf';

# Win95 reboot message to be executed if patches are applied
$win95RebootMsg = "$basePatchDir\\win95\\w95upd.exe";

# WinNT patch install method
$winntInstallPatch = '';

# WinNT patch install method parameter
$winntInstallPatchParm = 'update -z -u -q';

# WinNT reboot message to be executed if patches are applied
$winntRebootMsg = "start $basePatchDir\\winnt\\shutdown /l /y /r /t:30
                   \"Patches completed.  System rebooting.\"";

########################################################################

# First we must determine what OS we are running
($osString, $majorVer, $minorVer, $osBuild, $osId) = Win32::GetOSVersion ();
if ( $DEBUG >= 2 ) {
   print "ID = $osId\n";
}
# win32s
if ( $osId == 0 ) {
   print "Can't patch win32s!.";
   exit ( -1 );
}
# Windows 95
elsif ( $osId == 1 ) {
   # Print the OS information
   print "OS version is Windows 95$osString.\n";

   # Set the $patchDirectory
   $patchDirectory = "$basePatchDir\\win95\\$osBuild";
   $installPatch = $win95InstallPatch;
   $installPatchParameter = $win95InstallPatchParm;
   $rebootMessage = $win95RebootMsg;
}
# Windows NT
elsif ( $osId == 2 ) {
   # Needed by Windows NT EventLog
   require "NT.ph";


   # Print the OS information
   print "OS version is Windows NT $majorVer.$minorVer ";
   print "(build $osBuild: $osString)\n";
```

```
    $patchDirectory = "$basePatchDir\\winnt\\$osBuild";
    $installPatch = $winntInstallPatch;
    $installPatchParameter = $winntInstallPatchParm;
    $rebootMessage = $winntRebootMsg;

    $eventLog = 1;
    Win32::OpenEventLog ( $ntlog, $computer, $eventLogFile) || warn $!;
}
# Unknown
else {
    print "Unable to determine what OS we're running!\n";
    print "Script exiting...\n";
    exit ( -1 );
}

if ( $DEBUG >= 1 ) {
    print "Patch Directory = $patchDirectory\n";
    print "Install Patch method = $installPatch\n";
    print "Install Patch parameters = $installPatchParameter\n";
    print "Reboot Message = $rebootMessage\n";
}

# Now we are ready to gread the patch.list file
open ( PATCHLIST, "$patchDirectory/patch.list" ) || die $!;

# Now we loop through the listing of patches
while ( $patch = <PATCHLIST> ) {

    # Check for comment
    $tmpChar = substr ( $patch, 0, 1);
    if ( ("$tmpChar" ne ";") && ($tmpChar ne "#") ) {

        # Get the individual patch information
        chop ( $patch );
        ( $pRegKey, $pSourceDir ) = split ( /:/, $patch );

        # ...This should be able to handle other hives as well as HKLM...
        # ...a kludge for now...
        $pRegKey =~ s/HKLM\\//;
        $regObject = $HKEY_LOCAL_MACHINE;

        # If we can open the given registry key, then we assume that
        # the patch is in place and go on
        if ( $pRegKey ne "" ) {
            if ( $DEBUG >= 2 ) {
                print "Patch Registry Key = \n     $pRegKey\n";
                print "Patch Source Directory = \n     $pSourceDir\n";
            }
            if ( $regObject->Open ( "$pRegKey", $patchVer ) ) {
                print "Patch $pSourceDir already installed!\n";
            }
            else {
                $reboot = 1;
                print "Installing $pSourceDir...\n";
                $installTmp = $installPatch . $patchDirectory . "\\" .
                              $pSourceDir . "\\" . $installPatchParameter;
                print "$installTmp\n";
                system $installTmp;
                print "\n";
                if ( $eventLog == 1 ) {
                    if ( $regObject->Open ( "$pRegKey", $patchVer ) ) {
                        Win32::WriteEventLog ( $computer, $eventSource,
                              &EVENTLOG_SUCCESS, &NULL, 3002, &NULL, 0,
                              "$pSourceDir" );
                    }
                    else {
```

```
                    Win32::WriteEventLog ( $computer, $eventSource,
                           &EVENTLOG_ERROR_TYPE, &NULL, 3003, &NULL, 0,
                           "$pSourceDir" );
                 }
              }
           }
        }
     }
  }

  # Check to see if we installed patches
  if ( $reboot ) {
     system $rebootMessage;
     if ( $eventLog == 1 ) {
        # Event : rebooting the system
        Win32::WriteEventLog ( $computer, $eventSource,
               &EVENTLOG_INFORMATION_TYPE, &NULL, 3004, &NULL, 0, "" );
     }
  }

  if ( $eventLog == 1 ) {
     # Event : Successful script completion
     Win32::WriteEventLog ( $computer, $eventSource, &EVENTLOG_INFORMATION_TYPE,
                           &NULL, 3000, &NULL, 0, "" );
     Win32::CloseEventLog ($ ntlog );
  }

  exit (0);

  ########## end of patch32.pl ##########################################
  #######################################################################
```

Appendix C: Example patch.list

```
;
;     Patch listing for Windows NT 4.0 ( build 1381 )
;
;     Author          : Gerald ( Jerry ) Carter
;                       jerry@eng.auburn.edu
;                       College of Engineering Network Services
;                       Auburn University
;     Date Created    : February 20, 1998
;     Last Update     : June 1, 1998
;

; Service Pack 3
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\ServicePack3:sp3

; This fix corrects GETADMIN problem
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\Q146965:admnfix

; This fix corrects the chargen/telnet port issue (135)
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\Q154460:chargen

; Workaround for Pentium problem with invalid opcode
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\Q163852:pentfix

; Fix for newtear and bonk attacks.  Also ICMP attacks
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\Q179129:tearfix

; This fix is for lsahack.exe problem..  Also increased encryption on
; the LSA secrets in the registry.
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\Q184017:lsa2fix
```

Monitoring Utilization in an NT Workstation Lab

Paul Kranenburg
Erasmus University Rotterdam
e-mail: kranenburg@few.eur.nl

Abstract

This paper describes a set of tools used to monitor and maintain NT workstations at the faculty of Economics at the Erasmus University Rotterdam. These tools have been employed in different shapes and on a variety of systems since 1992. In particular, the data gathered on usage patterns of workstations in the faculty's public student labs has proved to be valuable for long-term planning of lab capacity and resource allocation. The collected data is also made available for live monitoring of workstation status and utilization. This is helpful both to tutors giving classes and to lab assistants while troubleshooting. Since mid-1997, all lab machines run Windows NT and the monitoring and maintenance system has been updated to take advantage of features that NT offers.

0. Introduction

As heirs from a predominantly do-it-your-self style of operation, contemporary PC desktop systems still present some challenges to any system administration group setting out to efficiently manage them as a uniform collection of computing resources. Only recently, with the arrival of Windows NT as a mainstream desktop operating system, a reasonably balanced platform has emerged that allows a decent form of system administration to take shape in this area. This paper addresses two issues which we consider to be of prime importance to sustaining smooth operation of our installed base of desktop workstations: (1) maintaining information on utilization and operational status of individual systems at a central location, and (2) the ability of making any necessary adjustments to the system configuration swiftly and with minimal disruption.

The tools presented here share a common design strategy, which in general takes the form of a small policy-less stub on each workstation that communicates with one or more counterparts running on a server. Such a scheme facilitates the concentration of all system configuration policies at a central location and also allows for easy processing and consultation of system status data collected on all desktop workstations.

1. The early stages

The control and monitoring features described here have their roots in the early 1990's, at which time we prepared to integrate a lab of DOS-based machines into an existing network environment consisting of Unix servers and workstations. Noteworthy characteristics of the lab environment include: (1) support for a large number (~10000) of users, demanding a docile authentication system; (2) all machines are directly connected to the Internet, also requiring proper authentication; (3) major changes can be made only once a year (during the summer break); (4) class schedules require a high degree of availability and robustness.

To compensate for the lack of any access control mechanisms both in the PC firmware and in DOS, we developed a set of utilities that ran as a front-end to DOS and which implemented authentication and access control by accessing (through a simple network protocol) a server-resident account database. Once in place this framework was easily extended to include the information to accurately track logon and logoff activity on the lab stations.

Note that the goal of this effort was to offer a reasonably stable environment where helpdesk operators and tutors can turn their attention to assisting the users, instead of needlessly worrying about basic system integrity. It was not designed to cover any deliberate attempt to circumvent the access control features.

A cost-effective method of low-level access control is achieved by plugging a PROM on the PC's network controller board. The PROM hooks into the system boot-strap process, allowing the default boot sequence to be replaced by one that effects an authentication transaction with a central accounts database implemented on a Unix server.

The PROM itself merely contains the necessary driver and networking code to retrieve a second-stage boot program (referred to here as the *PC monitor program*, analogous to what is commonly found built-in on professional workstations) using the TFTP protocol. This PC monitor program then implements the policies for authentication and access control by consulting a service running on a Unix host.

A simple UDP-based protocol is used to communicate with the server. All necessary network parameters that the PC monitor program needs are retrieved using the BOOTP protocol, including the address of the server running the authentication service which is implemented as a BOOTP 'vendor extension'.

The PC monitor program collects the user credentials and presents these to the server for authentication. If authentication is successful, the monitor program receives a set of capabilities based on the user-id, which are used to determine how the boot process can continue.

Normally, the only way to progress is to boot from the local disk holding the regular operating system. However, administrative accounts may receive the capability to boot the machine from, for example, a diskette station or a maintenance partition on the local disk. In any case, the monitor program reports its actions to another service process for the purpose of maintaining login session statistics. As soon as the monitor program is loaded a 'boot' event is generated that marks the time of a machine start. A 'login' event is generated when a user is successfully authenticated and has selected a valid device or partition to boot from. These events allow the server to maintain a history of login activity on all lab stations, similar to the UTMP records that can be found on Unix systems. This history of login sessions can be effectively used to generate both instantaneous and long-term usage statistics.

2. Transition to NT workstation

Early 1997 we began to prepare for the installation of Windows NT Workstation on all lab stations.

It was deemed desirable to continue to operate in some form the UTMP system that had already proved to be a useful tool in the pre-NT era. Indeed, the system's multi-tasking features open the prospect of extending its abilities by continually monitoring and reporting on the system status instead of being active only as a front-end to the operating system.

Though before undertaking the porting job, several other constraints were to be taken into consideration. Whereas the time of introduction of NT (driven by class timetables) in the student labs was set for the summer of 1997, our network server configuration remained locked into supporting existing PCs at other departments for some time to come, precluding any radical changes in our network server setup. The network servers all run Novell's NetWare 4 operating system and account management is based on the NetWare Directory Services (NDS).

Thus, the Windows NT installation must co-operate with the existing server network installation using Novell's network client software for Windows NT, including automatic local user account management based on the NDS database.

It follows that the envisioned port of the UTMP session logging mechanism had to fit into this hybrid environment. Fortunately, implementing UTMP on top of NT's audit-event capabilities, as explained in the next section, nicely de-couples it from the lower-level authentication packages employed by NT.

Another important consideration is the organization of workstation software maintenance. Our goal is to avoid any manual intervention to install or upgrade software components. In particular, there should be no need for anyone to be logged on to the console for these installations or upgrades to be accomplished. In addition, any mechanism deployed for this purpose must be able to strictly separate machine-specific and user-specific parts of the installation process.

Since none of the commercial solutions we explored in early 1997 (e.g. McAfee Brightworks and Microsoft SMS v1.1) met the demands posed by the hybrid environment as sketched above, we decided to develop our own. The results of this project are discussed in section 5.

3. The NT utmp service

Since NT offers a mature operating system environment on the lab stations, many functions of the original PC monitor program became obsolete. For instance, access to the local machine is now controlled by NT's security subsystem, assisted by a NetWare component that is responsible for authentication using the NetWare directory services; obviating the home-grown authentication mechanism developed for DOS-based installations.

The monitor program is still retained however to control the boot process. As before, it only allows booting from the device holding the default operating system unless proper credentials are presented that enable other bootable media for maintenance purposes.

Since the PC monitor program is no longer in a position to obtain the information to generate UTMP records, this functionality has been moved into an NT service program. The operation of this service relies on the system's built-in auditing features. Several groups of security-related facilities can be independently enabled to generate *security audit events*. These events are collected by the system in the Security Event Log, and can be examined by a properly privileged process. A description of the general framework for event logging can be found in the NT Resource kit[1]; programmatic details on accessing the system event logs are given in the MSDN documentation[2]. The UTMP service uses the LOGON/LOGOFF category of auditing events that are generated by the so called Authentication Packages, such as the Graphical Identification and Authentication modules (GINAs) and SMB network connection authenticator (showing up in the event logs as the KsecDD module).

Within a given category each event is uniquely identified by its event-id. The UTMP service looks for just two types of events in the Logon/Logoff category: SUCCESSFUL LOGON and SUCCESSFUL LOGOFF. Associated with each event type is a collection of additional data, the interpretation of which is specific to the event-id.

The LOGON event carries the following event-specific data:[1]

- the user name,
- the domain name to which the user belongs,
- a logon id, which is a unique identifier for each logon session,
- logon type (e.g. *interactive*, *batch*, *service*, etc.),
- a logon process,
- the name of the authentication package,
- the name of the remote machine (if any) where the session originates.

The LOGOFF event carries these additional data:

- user name,
- the domain name to which the user belongs,
- the logon id, the unique identifier for the logon session,
- the logon type.

The UTMP service program utilizes this data to keep track of logon sessions on the machine. In particular, it uses the <logon id> that is passed along with each LOGON and LOGOFF audit event to pair the events that mark the start and end of logon sessions.

In addition to the audit events in the LOGON/LOGOFF category, the NT system events CTRL_LOGOFF_EVENT and CTRL_SHUTDOWN_EVENT, which are generated when a logon session on the console is terminated, are also monitored. This is done to safeguard against latencies in the generation of the LOGOFF audit events

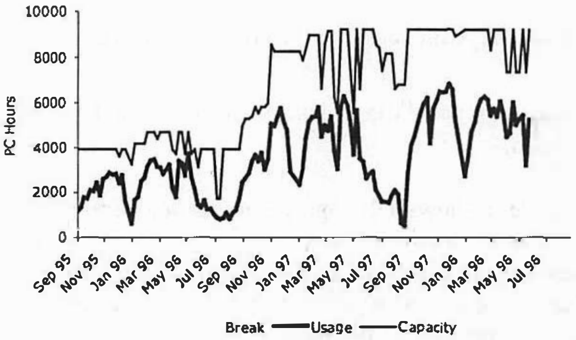


Figure 1: available capacity vs. usage spanning three academic seasons

as a result of inadequate behaviour of some NT authentication packages[2].

A useful bonus of the fact that the UTMP service tracks logon sessions, is the ability to run an arbitrary program in the context of the local system account at the start or end of a user logon session. This feature allows us to apply and cleanup user-specific modifications to the local machine configuration in support

[1] This additional data can be easily viewed using the EventViewer utility that comes with Windows NT.

[2] Apparently, the system generates a LOGOFF audit event only when the last reference to a process in a logon session goes away. Hence, neglecting to close, say, a process or thread handle to any of the user processes postpones the LOGOFF event even though the console may long be vacated. This behaviour was exhibited by an early version of the NetWare logon module (NWGINA). It was also observed in the telnet service that comes with the Windows NT 4.0 resource kit.

of a number of ignorant applications without having to relax standard security levels.

4. Displaying session data

The data collected by the UTMP daemon enables us to produce both long-term and short-term overviews of the usage of all stations in the labs. Long-term overviews provide valuable information to spot growing capacity problems well in advance. The figures illustrate aggregated usage data collected over a three-year period.

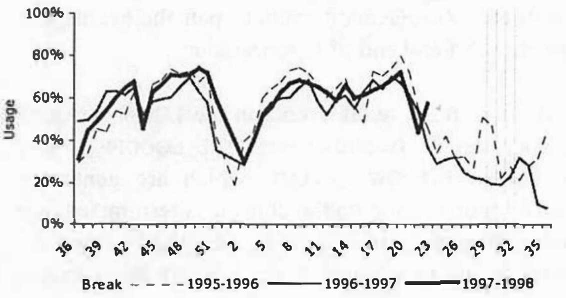


Figure 2: relative utilization in three successive seasons.

Figure 1 shows lab capacity utilization during three academic seasons based on weekly averages. Lab capacity is expressed in *PC hours* which is the combined login session time available on all machines - i.e. the product of the number of lab stations and lab opening hours. The actual capacity in a given week may vary due to holidays and scheduled maintenance. The bold line shows the measured utilization from the UTMP data. The shaded areas represented the midsummer holiday breaks.

Figure 2 illustrates the relative utilization for three academic seasons. It shows that the degree of utilization during each period remains the same despite a marked increase in capacity realized in November 1996. From this observation one may conclude that "demand" for lab capacity still exceeds the "supply". It should be noted that - in our experience - a weekly average utilization degree exceeding 70% actually means "a very busy period".

Lab station usage is also converted in real-time to a set of HTML pages that can then be viewed on any workstation running an HTML browser. The information plotted in these pages includes details about all lab stations known to the UTMP daemon, including the currently logged-on username and the duration of the logon session.

An example of a real-time view can be seen in figure 3. The map layout reflects the actual locations of the workstations in the lab rooms. Color codes are used to give an indication of each station's disposition, for example `idle', `in use' or `not responding', making it easy for support personnel to spot signs of trouble at the earliest opportunity. The data is also displayed on a *kiosk* monitor which arriving students can use to quickly locate an available workstation in one of the lab rooms.

5. The Autoadm service

The development of the mechanism to automatically distribute software and system patches quickly turned into an exercise in glueing together various existing and readily available tools. Things are set in motion at the workstation by an NT service program called *autoadm*[3].

Its sole purpose is to periodically set up a network connection to a central repository containing the full set of tools and data that define the distribution of software in units called *packages*. A package can be anything, ranging from the complete installation of MS Office to adjusting the size of the system's swap file. Note that this service neither places any restrictions on nor offers assistance with the construction of the package contents. However, a simple interface for passing status information and error messages is defined, which all package installation scripts must adhere to. A frequently used tool at our site for actually constructing package installation procedures suitable for unattended installation is McAfee Wincompare.

At the heart of the distribution mechanism is a Perl[3] script, that is started once the network connection to the repository has been set up successfully. This script consults a package configuration file, that defines the packages that are to be installed on the workstation. The package configuration file allows various control parameters to be specified that determine where and when a particular package should be installed. For example, an arbitrary collection of machines can be named as a group that can be used as optional per-package target parameter in the configuration file.

[3] The installation of this service is integrated with the initial installation of NT, so its services are available right away without further operator intervention.

The results of an attempt to install a package are recorded in two locations: locally in the machine's system registry and remotely by opening a network connection to a service process designed to assist the NT package installation procedure.

This service process controls the package installation in several ways:

(1) Establishing the workstation's identity by requesting its unique installation key and verifying it by engaging in a simple challenge/reply using a pair of cryptographic keys assigned to each workstation[4].

(2) Offering a load-balancing option to control the pace when installing large numbers of machines concurrently.

(3) Maintaining a centralized logging facility through which it is easy to monitor the progression of package installations on all workstations.

(4) Providing a secure channel on which auxiliary (presumably sensitive) package installation parameters can be transported, deploying the same cryptographic keys that are used in the authentication step in (1). For instance, we use this facility to periodically change the local `Administrator` password on all NT machines.

The central repository is located on a Unix host running Samba[4], which makes it easy to connect to using NT's native SMB protocol. Thus, the service is functional immediately after the initial NT installation is complete and will automatically extend the operating system installation by processing the packages prepared on the distribution repository.

So far, the description of the automatic package distribution mechanism pertains to packages targeted at machines. While most packages are machine-specific and indeed are scheduled to run at a point of time when no one is logged on to the machine, it is sometimes necessary to effect accompanying changes in the user's context. A completely analogous method is used to distribute and administer such user-specific packages. In fact, the same package configuration file format and perl script is used to accomplish the installation of user-specific packages. The distinction between user and machine packages is entirely the result of the different environment in which the package distribution tools operate, which can be summarized thus: (1) the package update script runs as part of the user's login script; (2) a different - user accessible - package repository is used to hold the package data; (3) package installations are registered in the user portion of the NT registry; (4) the results are logged in a separate log file on the server.

[4] Assignment of the unique identifier and the cryptographic keys are also part of the initial installation procedure.

6. The future

The package installation mechanism was born out of necessity to fit the parameters of our environment. It shows that combining the strengths of several proven and readily available tools can achieve a marked improvement in tractability of a large desktop network. A weakness of the current system can be found in the handling of the ever-increasing amount of logged data. Obviously, employing a mature database system would help out in that area. Hence, we'll continue to look at off-the-shelf solutions as they arrive at the marketplace.

Likewise, the data generated by the UTMP service is also in need of a more sophisticated data management system. The accumulation of login session data spanning several years is starting to stretch the capability of the currently used tools to process the collected information efficiently. This topic will be addressed in a future revision.

7. References

1. Windows NT workstation 4.0 Resource kit; Microsoft Press.

2. Microsoft Developer Network Library; Microsoft Corporation

3. Larry Wall, Tom Christiansen & Randal L. Schwartz, Programming Perl; O'Reilly 1996

4. SAMBA; http://samba.anu.edu.au/samba/samba.html

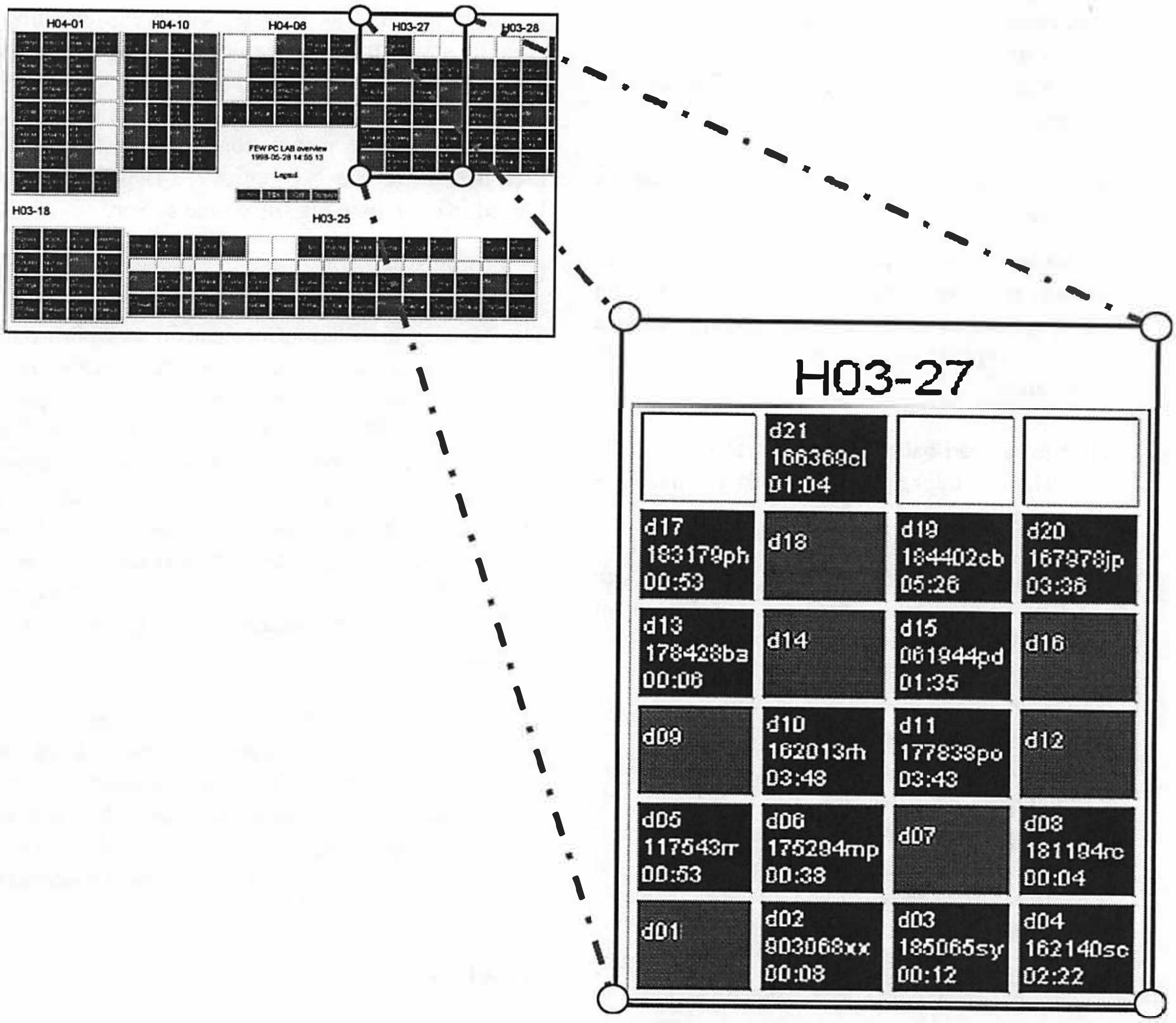


Figure 3: browser view snapshot of real-time utilization

File System Security: Secure Network Data Sharing for NT and UNIX

Bridget Allison - bridget@netapp.com, Robert Hawley, hawleyr@netapp.com,
Andrea Borr - aborr@netapp.comMark Muhlestein - mmm@netapp.com
David Hitz - hitz@netapp.com
Network Appliance Inc.

Abstract

Sharing network data between UNIX and NT systems is becoming increasingly important as NT moves into areas previously serviced entirely by UNIX. One difficulty in sharing data between UNIX and NT is that their file system security models are quite different. NT file servers use access control lists (ACLs) that allow permissions to be specified for an arbitrary number of users and groups, while UNIX NFS servers use traditional UNIX permissions that provide control only for owner, group, and other. This paper describes a merged model in which a single file system can contain both files with NT-style ACLs and files with UNIX-style permissions. For native file service requests (NFS requests to UNIX-style files and NT requests to NT-style files) the security model exactly matches a UNIX or NT fileserver. For non-native requests, heuristics allow a reasonable level of access without compromising the security guarantees of the native model.

Introduction

For the past 18 months Network Appliance has been shipping its Multiprotocol filer - a dedicated file server appliance which can be licensed and configured to speak some combination of NFS, CIFS and HTTP. One of the primary challenges facing any solution which offers a single location for data to clients that speak disparate protocols is the maintenance of data integrity. If the file server cannot ensure the integrity of the data then it has failed in its purpose. When a client trusts a file server with its data that leap of faith must be honored. If it is not, then the file server will not be used for data storage. Clearly a dedicated file server appliance must address data integrity head-on if it is to fulfill its mission in life. If we set aside the hardware issues related to data integrity and focus only on the software issues then data integrity is comprised of two parts: secure file/byte range locking and file and directory level security.

This paper addresses Network Appliance's approach to the second part of the data integrity story: file level security.[1] Specifically, it describes how Network Appliance has merged UNIX and Windows NT (NTFS) file level security[2] to provide a flexible yet secure merged security model which allows Windows and NFS users to use the most appropriate security model for the data being stored. This model allows all data to be accessible by all protocols, UNIX perms and NT ACLs accommodated within the same file system while giving the administrator control over which file level security model will be used in each area of the file server.

This paper also describes how the merged security model is being used by a Network Appliance customer: Glaxo Wellcome R&D.

Finally, this paper details areas to be considered when migrating to a merged system and mentions other tools that may be useful in a mixed UNIX and Windows environment. It also talks briefly about how other multiprotocol solutions address merged security issues; specifically AT & T's Advanced Server for Unix and Samba.

A word about qtrees

Before going into the design of the integrated security model it is worth taking a moment to explain how the NetApp filer uses qtrees since they are the basis for security style configuration. On a NetApp filer qtrees allow a flexible segmentation of a filesystem for quota management, opportunistic lock (oplock) configuration, backup management and also security styles. Since a qtree can be dynamically grown, deleted or created this provides a lot of flexibility for the administrator. Qtree

---

[1]For a thorough treatment of how Network Appliance has addressed the file locking part of data integrity please read Andrea Borr's paper:
'SecureShare: Safe UNIX/Windows File Sharing through Multiprotocol Locking' to be presented at the USENIX NT Symposium 98.

[2] The integrated security functionality (Windows NT ACLs support integrated with UNIX perms) is available in Data ONTAP 5.1 from Network Appliance Inc.

security styles can be changed on the fly, conversion of a UNIX style qtree into an NTFS style qtree for example, can be done quickly and easily as your storage needs change.

For a simpler configuration, an entire volume can itself be a qtree. So, for example, setting the security style on the root of a single volume filer with the command

```
qtree security / ntfs
```

causes the volume to report a filesystem type of 'NTFS' to Windows clients and allows NT ACLs to be created across the entire filesystem. This is the equivalent of formatting the C: drive of your Windows NT file server as NTFS from a file security perspective.

1. Merged Security Model Design Goals

The Network Appliance merged security model was designed to meet several goals:

(1) Make Windows Users Happy

Support an NT-centric security model based on NT Access Control Lists (ACLs). To a Win95/98 or NT client using CIFS (the standard Windows remote file service protocol) this security model should behave exactly like a Windows NT file server.

(2) Make UNIX Users Happy.

Support a UNIX-centric security model based on UNIX permissions. To an NFS client, this security model should behave exactly like a UNIX NFS server.

(3) Let Them Work Together.

Provide reasonable heuristics so that Windows users can access files with UNIX-centric security, and UNIX users can access NT-centric files.

To meet these goals, NetApp supports both NTFS-trees and UNIX-trees. Administrators can set the security model on the root of a WAFL[3] file system, but they can also set the security for individual qtrees, allowing Windows and UNIX users both to be happy in the same file system.

[3] Write Anywhere File Layout - the Network Appliance filesystem

Native requests -- NFS to a UNIX-tree or CIFS to an NTFS-tree -- work exactly as expected. UNIX-trees are modeled after Solaris, and NTFS-trees are modeled after Windows NT. Non-native requests use heuristics designed to operate as intuitively as possible while still maintaining security.

CIFS requests to UNIX-trees are handled by mapping each CIFS user to an equivalent UNIX user, and then validating against the standard UNIX permissions.

In the simplest case, a user named John might have the account "john" on both NT and UNIX. When John starts a CIFS session, the filer looks up "john" in /etc/passwd (or over NIS), and uses the specified UID and GID for all access validation.
A user name mapping file handles the case where John's NT account is "john", but his UNIX account is "jsmith". NT users with no UNIX account may be mapped to a specified UNIX account, or they may simply be denied access.

NFS requests to NTFS-trees in phase 1 of the implementation are validated using special UNIX permissions that are set whenever an ACL is updated. The UNIX permissions are guaranteed to be more restrictive than the ACL, which means that users can never circumvent ACL-based security by coming in through NFS. On the other hand, since UNIX permissions are less rich than NT ACLs, a multiprotocol user may be unable to access some files over NFS even though they are accessible over CIFS. In practice, NFS access to NTFS-trees works well for the owner, and for access granted to the NT "everyone" account, but not for other cases.

In a future addition of the integrated security model we will implement UID to SID mapping which will allow us to map each NFS user to an equivalent CIFS user, and then validate against the file's ACL. At that point it may not be useful to maintain a set of UNIX perms on a file which has an ACL.

In addition to UNIX-trees and NTFS-trees, the filer supports a Mixed-tree in which the security model is determined on a file-by-file basis. Files created by UNIX users use UNIX permissions, and files created by NT users use NT ACLs. A file's security model may be flipped from one style to another by NFS "setattribute" or CIFS "set ACL" requests. Only a file's owner can flip the style.

Several features allow special control over privileged users. The filer supports the NT "Administrators" local

group, which lists the NT accounts that have administrator privileges. The NT User Manager interface can be used to manage the "Administrators" local group over the network. The user name mapping file may be used to map the NT "Administrator" account into UNIX "root", or to a non-privileged UNIX account such as "nobody". Privileges for UNIX "root" are controlled using the /etc/exports "root=" flag. Requests from "root" are mapped to "nobody" unless the "root=" flag on an export explicitly allows root privileges.

2. Understanding File System Security Models

2.1 Native File System Security Models for NT and UNIX

This section provides a quick overview of UNIX and NT file system security, since many people are familiar with one or the other, but not both.

2.1.1 UNIX Security Model

UNIX uses user IDs (UIDs) to identify users, and group IDs (GIDs) to identify groups. The UNIX permissions stored with each file consist of:

- UID of the owner
- GID of the owner
- User perm bits (defining read, write, execute for the owner)
- Group perm bits (defining rwx for the group)
- Other perm bits (defining rwx for anyone else)

When performing validation, UNIX first determines whether the request is from the file's owner, someone in the file's group, or anyone else, and then uses the user perms, group perms, or other perms, respectively.

2.1.2 NT Security Model

NT uses security IDs (SIDs) to identify both users and groups. The NT permissions for each file consist of:
- SID of the owner
- SID of the owner's primary group
- ACL (Access Control List) for the file

The ACL contains one or more access control entries (ACEs). Each ACE contains a SID, indicating the user or group to which the ACE applies, and a set of permission bits. NT permission bits include the three UNIX bits -- read, write, and execute -- as well as "change permissions" (P), "take ownership" (O), "delete" (D), and others. An ACE can either allow the specified permissions, or deny them. When performing validation, NT walks the list of permissions granted to the user by the ACEs until a deny (or the end of the list) is reached. At that point processing stops.

UNIX/NFS and NT/CIFS also differ in how they authenticate users. NFS is a connectionless protocol, and each NFS request includes the UID and GIDs of the user making the request. The UNIX client determines the UIDs and GIDs when the user first logs in, by looking at the files /etc/passwd and /etc/groups. CIFS is session based, so the identity of the user can be determined just once, when the session is first set up. At session connect time, the client sends the user's login name and password to the CIFS server, and the server determines the sessions user SID and group SIDs. Servers in an NT domain environment commonly forward the name, the challenge and the client's response to an NT domain controller (DC) and let the DC determine the SIDs.

2.2 Important Issues for Non-Native Filesystem Security

The previous section describes the native security models for both UNIX and NT. This section examines the issues that are important for non-native security -- the security for an NFS request to a file with NT ACLs, or a CIFS request to a file with UNIX permissions.

The primary function of any filesystem security model is to validate requests -- to accept them or deny them based on the authenticated user and the permissions of the file. Thus, validating non-native requests is obviously an important topic.

In addition, there are several file system actions that require special attention. In particular, to define how non-native requests are processed, we must answer the following questions:

- How are requests to display permissions handled?
- How are requests to set permissions handled?
- How are permissions set for newly created files?

Handling non-native permissions is tricky, because the NT and UNIX permission models are so different. To define a merged security model, one must specify how an NT ACL will be converted to UNIX permissions, and visa-versa.

3. Handling Non-Native CIFS Requests

This section describes how we handle CIFS requests to UNIX-style files. Remember that files with UNIX permissions occur both in UNIX-trees, where all files are UNIX-style, and in Mixed-trees, which have both UNIX-style and NTFS-style files.

Section 3.1 discusses how non-native CIFS requests are validated, and section 3.2 discusses the non-native handling for displaying permissions, setting permissions, and creating new files.

3.1 Validating Non-Native CIFS Requests

NetApp filers validate CIFS requests to UNIX-style files by generating a mapped UID (and GIDs) for each CIFS session, and then using the UID (and GIDs) to check against the UNIX permissions.

Suppose that the NT user "john" connects to a filer. Here are the steps that the filer will take to determine the mapped UID and GIDs for "john".

(1) The filer sends a request to the NT domain controller (DC), to authenticate "john", and to find out the NT SID for "john".

(2) The filer looks in the user mapping file to determine whether the NT account "john" maps into a different account name under UNIX. In this case, let's assume that "john" maps into the UNIX account "jsmith".

(3) The filer looks up "jsmith" in /etc/passwd (possibly via NIS) to determine the UNIX UID for John.

(4) The filer uses /etc/groups (possibly via NIS) to determine the UNIX GIDs for John.

These steps provide each CIFS session with a full set of UNIX authentication information, which allows the filer to easily validate most requests against the UNIX permissions.

Some CIFS operations don't map well to UNIX operations, so they must be handled specially:

- Set ACL

The CIFS "set ACL" operation is always denied in UNIX-trees. In Mixed-trees, a "set ACL" operation is only allowed by the owner (i.e. the mapped UID for the CIFS session must match the file's UID.) In this case, the file is converted from UNIX-style permissions to NT-style permissions.
Only the owner can set an ACL, because in UNIX only the owner is allowed to set attributes.

- Take Ownership

The CIFS "take ownership" operation is always denied in UNIX-trees. In Mixed-trees, only the file's owner can "take ownership". Like the "set ACL" request, this converts the file to NT-style permissions.

3.2 Request Processing for Non-Native CIFS Requests

This section considers non-native CIFS requests in light of the three questions listed above, in Section 2.2, "Important Issues for Non-Native Filesystem Security":

- How are requests to display permissions handled?
- How are requests to set permissions handled?
- How are requests to create a file handled?

3.2.1 How are requests to display permissions handled?

For non-native CIFS requests to display permissions, WAFL dynamically builds an ACL designed to represent the UNIX permission as best as possible.

On might hope to build an NT ACL that perfectly represents the UNIX permission like this:

Owner - map the file's UID into an NT SID.
Group - map the file's GID into an NT SID.
ACL
ACE for owner SID, based on UNIX user perms.
ACE for group SID, based on UNIX group perms.
ACE for well known NT "Everyone" SID, based on UNIX other perms.

In practice, this doesn't work if we have no way to convert UIDs or GIDs into SIDs. In constructing the ACL, we can only use well-known NT SIDs, and the SID for the CIFS session itself. These are sufficient to let us construct an ACL that, while not perfect, does provide useful information.

Each ACL contains two access control entries (ACEs):

- An entry for NT "Everyone" SID, based on the UNIX "other" permissions.

- An entry for the SID of the CIFS session, based on whichever UNIX permission is appropriate. If the mapped UID is the file's owner, the ACE is based on the UNIX owner perms. If the group matches, then it's based on the group perms. Otherwise it's based on the other perms.

If the CIFS session owns the file, then in the faked up ACL the session's SID is shown as the owner. If not, then the well known NT SID "CREATOR_OWNER" is shown as the owner.

3.2.2 How are requests to set permissions handled?

In UNIX-trees, CIFS requests to set permissions are always rejected[4]. Outside of UNIX-trees, non-native requests to set permissions are allowed only by the file's owner. If allowed, the ACL takes effect just as it would have if the file had been an NT-style file. After the set ACL request is processed, the file becomes an NT-style file.

3.2.3 How are permissions set for newly created files?

Unlike UNIX, which passes the permissions for the new file as part of the create request, NT expects permissions to be inherited from the parent directory.

To handle CIFS create requests in a UNIX-style tree, the filer simply inherits the parent directory's UNIX permissions:

- The file's owner is set to the mapped UID of the CIFS session.

- The file's group is set to the mapped GID for the CIFS session.

- The permission bits are inherited directly from the parent directory, except that SUID and SGID bits are cleared for non-directory creates.

4. Handling Non-Native NFS Requests

This section describes how we handle NFS requests to NTFS-style files -- i.e. files with NT ACLs.

[4] Although NetApp provides a client tool (SecureShare™ Access) that allows Windows users to set/view the UNIX perms on a file.

Section 4.1 discusses how non-native CIFS requests are validated, and section 4.2 discusses the non-native handling for displaying permissions, setting permissions, and creating new files.

4.1 Validating Non-Native NFS Requests

Whenever an NT ACL is set or changed, WAFL calculates a corresponding set of UNIX permissions. As a result, very little special processing is required to validate NFS requests to files with NT ACLs. Simply doing the normal checks against the UNIX permissions does (almost) the right thing. The rest of this section describes how the UNIX permissions are constructed from the ACL, and explains the "(almost)" in the previous sentence.

Converting an NT ACL into UNIX permissions is surprisingly tricky. This section gives a brief overview, but an observant user may encounter slight differences in the actual implementation.

- The file's UID is set to the mapped UID for the CIFS session.

- The file's GID is set to the mapped GID for the CIFS session.

- The UNIX user perm is set based on the access rights that the ACL grants to the CIFS session creating the file. (These may or may not be explicitly specified by an ACE for the owner.)

- The UNIX other perm is set based on the access rights granted to the NT "Everyone" account. (If the ACL contains any denies, then the denied permissions are subtracted from the other perms.)

- The UNIX group perm is set equal to the other perm.

This design avoids security holes by ensuring that the UNIX permission is always at least as restrictive as the NT ACL is. Unfortunately, UNIX permissions cannot represent the full richness of the NT security model. As a result, a file that a user can reach via CIFS may not be accessible via NFS.

Because NT supports some specific permissions that UNIX lacks, it is not possible to rely entirely on the UNIX permissions to validate NFS requests:

REMOVE/RMDIR

Only the owner of a file is allowed to delete it. This is necessary to avoid violating the NT "delete child" permission.

CREATE
Only the owner of a directory can create anything in it. This is required in order for NT ACL inheritance to work properly.

4.2 Request Processing for Non-Native NFS Requests

This section considers non-native NFS requests in light of the three questions listed above, in Section 2.2, "Important Issues for Non-Native Filesystem Security":

- How are requests to display permissions handled?
- How are requests to set permissions handled?
- How are permissions set for newly created files?

4.2.1 How are requests to display permissions handled?

As described above, in section 4.1, every file with an NT ACL also has a set of UNIX permissions stored with it. To handle an NFS "get attributes" request, the filer simply returns those stored permissions.

4.2.2 How are requests to set permissions handled?

In NTFS-trees, NFS requests to set permissions are always rejected.

In Mixed-qtrees, only a file's owner is allowed to set permissions. When UNIX permissions are set on a file, the NT ACL is deleted; the file changes from NT-style to UNIX-style.

Note that "set attribute" requests that update non-security information such as access time or modify time are allowed even in NTFS-trees, and they do not delete the NT ACL.

4.2.3 How are permissions set for newly created files?

NFS creates in NTFS-trees are only allowed by the directory's owner. This is because an NT SID is required to handle NT ACL inheritance.
An NFS request has no SID, but for a create request from a directory's owner, WAFL can use the owning SID from the directory's ACL and handle ACL inheritance according to normal NT rules. (If the parent directory has no ACL, WAFL follows normal UNIX rules.)

In Mixed-trees, NFS create requests are handled according to normal UNIX rules.

5. Other Issues

5.1 FAT versus NTFS

NT servers support both FAT file systems and NTFS file systems. FAT is the traditional DOS file system -- it has no file-level security at all. The NTFS file system was designed for NT, and it supports a security model based on ACLs.

Since NTFS-trees and Mixed-trees both support ACLs, they must be advertised as "NTFS" file systems. (If they were advertised as "FAT", clients would assume that they had no ACLs, and would disable the interfaces for controlling ACLs.)

It is less obvious how to advertise UNIX-trees. One can make a case either way:

- UNIX-trees don't support ACLs, so advertising them as FAT sends a clear message to clients not to try to use ACLs. Advertising as NTFS would be confusing, since no ACLs are really present and any request to set ACLs will fail.

- UNIX-trees support file level security, and advertising them as NTFS allows the filer to display the UNIX permissions using faked-up ACLs. Advertising as FAT would be confusing, because it would seem to imply that no file-level security is present.

In the end we decided to advertise UNIX-trees as FAT, because this seems least likely to confuse Windows programs that absolutely must have ACLs.

Still, there are several situations in which it is useful to construct a fake ACL for an NT-style file, as described in section 3.2.1:

(1) In Mixed-trees

Mixed-trees contain files both UNIX-style and NTFS-style files. To support ACLs they must be advertised as "NTFS", yet not all files in them contain ACLs.

(2) In NTFS-trees that originated as UNIX or Mixed-trees

A qtree's security style can be changed at any time, so a tree that began as UNIX-style may later be converted to NTFS. In this case, it will clearly be advertised as "NTFS", but it may contain files without ACLs.

(3) In UNIX-trees accessed via an NTFS or Mixed Share.

The root of a filesystem may have NTFS or Mixed security, but it may contain a UNIX quota tree. In this case, the C$ (or root) share will be advertised as "NTFS", but there is nothing to stop a user from going down into the UNIX qtree and trying to display an ACL.

5.2 Migration from previous versions of Data ONTAP

For sites already using filers with CIFS, migration to the NTFS security model is an important issue.

System administrators can update the security model for any qtree (including the root of the filesystem), using the qtree command. The syntax is:

qtree security <quota tree> [unix|ntfs|mixed]

When a UNIX-tree is converted to an NTFS-tree, shares are advertised as "NTFS" instead of "FAT", and ACLs will be dynamically created for the files as described above in section 3.

When an NTFS-tree is converted to a UNIX-tree, shares are advertised as "FAT" instead of "NTFS", and any ACLs in the tree are simply ignored. ACLs are not actually deleted however, so if the tree is converted back to NTFS, the ACLs will still be present. The best way to delete the ACLs is to write a script that runs as root and chowns each file to its existing owner. (Remember that doing a chown or chmod deletes the ACL on a file.)

5.3 Share Level ACLs

Share level ACLs are also based on NT SIDs, and they can be edited over the network using the NT Server Manager.

For backward compatibility, share level ACLs based on UNIX user names continue to function, although they cannot be controlled via Server Manager.

6. User input into the design

This section describes some of the feedback we received from users during a customer conference on the design. It also describes how our design was changed to incorporate the feedback.

In February 98 Network Appliance invited a group of customers to a day of discussion around the integrated security design. We particularly wanted to focus on the integration issues that arose and the tradeoffs we'd made. We'd all convinced ourselves that our design was sound but 20 pairs of eyes all focused on just how this new technology would play out in their environment back home would be the reality check we needed.

We had identified one major gap in the phase 1 implementation - the lack of UID to SID mapping - and we were also interested in testing our logic regarding group mapping. In theory, the concept of mapping non-native access into a native context at the user level and then applying the security appropriate to the native user completely removes the need for group mapping. After some research with customers we had concluded that the vast majority of mixed UNIX/NT environments did not have identical group names in NT domains and NIS (e.g. NIS group 'eng' vs. NT global group 'Engineering Users'). So, if the names were not identical, then the best of all possible worlds would remove the need to maintain a group mapping file while also providing seamless mapping. Our answer was to map at the user level and having, for example, mapped a UNIX user into an NT user, apply the group membership of that NT user to the access request.

The UID to SID mapping, already identified as a possible next step, was a hot topic for discussion. Most of the customer representatives felt that for many areas within their organizations the security model would function just fine without UID to SID mapping. Typically, this phase 1 implementation was judged to be sufficient for areas where security was necessary without being too complex. Without UID to SID mapping a file with an ACL could be accessed via NFS by the file owner flawlessly but NFS access requests that came from someone other than the file owner would be evaluated based on the existence of the NT group 'Everyone' in the ACL. If there was no access granted to the group 'Everyone' then the NFS access would be denied. The lack of UID to SID mapping also prevents the ability to update the ACL from NFS: something that our customers requested we consider for a follow-up release. The discussion provided Network Appliance

with a clear picture of what the phase 2 implementation needed - UID to SID mapping was clearly necessary based on the feedback we got.

The decision to avoid group mapping was hotly debated and is being considered for the phase 2 integrated security implementation. Clearly, UID to SID mapping fixes a lot of problems and Network Appliance intends to invite customers back to review our revised design and preliminary implementation of this which will be based on feedback from phase 1 usage.

7. How the design works in practice

This section focuses on the real world experience of using the merged security model and gives examples of how one customer has successfully used the different security styles to share data in a mixed UNIX/NT environment.

Glaxo Wellcome, a UK-based Network Appliance customer, has been using filers for some time in a predominantly NFS environment, waiting for key Windows features before moving to consolidate mixed data on the filers. They spent some time testing the Network Appliance Data ONTAP 5.1 release, evaluating its usefulness in their specific environment. They are migrating from a VMS/Pathworks environment to using Windows NT and UNIX servers and are looking to offer users the same access to shared data with the same granularity of file level permissions which the VMS/Pathworks combination gave them.

One such group of users have moved from the VMS/Pathworks environment. Data is exported from a UNIX based Oracle database and is then analyzed by a NT based application., giving users a graphical interface to Pathology data, helping to more readily identify trends in the data.

The filers are in the Glaxo Wellcome Research division, an environment with UNIX, Windows NT and Windows 95 clients. Both UNIX and PC clients generate data which then has to be available to others. The PC clients run Microsoft Office applications to generate files that are then available to UNIX applications which incorporate this data into reports and web pages. The UNIX clients export data from Oracle databases into flat files which are then picked up by PCs and incorporated into technical reports, or analyzed by Windows based applications.

The Glaxo Wellcome requirement for a common data location that allows all users to access data independent of client platform is becoming increasingly common.

The Glaxo Wellcome R&D environment is rapidly migrating to a mixed UNIX/NT one, where Unix provides the database engine, and NT the reporting and GUI tools. The filer gives interoperability between the two platforms, without the database engine having to perform unnecessary file sharing operations via Samba, or similar. In addition, by making UNIX data directly available to NT users, those users are able to analyze the data using the Microsoft programming tools, with which they are familiar.

Since the Glaxo Wellcome R&D users were used to using VMS ACLs the ability to provide NT ACLs was an important piece of the solution. Providing Windows users with familiar ACLs also cuts down on calls to the HelpDesk since system administrators no longer have to explain UNIX file perms to Windows users. The system administrators see the filers as a simple solution to the complex problem of disk usage. Glaxo Wellcome uses Solaris, HP-UX and SGI IRIX plus Windows NT running on Compaq and HP servers. As usage of these servers changes the proprietary disks attached to the servers are left unused. Being able to host data on one platform gives much more flexibility during migration and makes better use of the investment in disks.

8. Real-world migration issues

This section covers some of the migration issues involved in moving from a UNIX security model to a merged NT/UNIX security model and describe some of the tools users need to ease the transition process. It also discusses some of the utilities that are useful but are beyond the scope of a pure fileserver appliance: account synchronization tools and authentication modules such as Nigel Williams' NIS GINA for NT.

8.1 Conversion from UNIX permissions to NT ACLs

For those who decide to move from a single file level security model to a merged environment the first question usually asked is 'Will I lose all my UNIX perms on those areas where I now need NT ACLs?' No one wants to ask their users to go in and re-set file permissions to take advantage of the new functionality. We realized early on in the design process that the need to fake up an ACL from a set of UNIX perms would be an important one, particularly during migration. Since we had many Windows customers using UNIX file security we

toyed with a one-time conversion tool which would read the UNIX perms on every file and set an ACL, using a mapping file which the administrator would set up in advance. This idea was popular during the customer discussions but no one wanted to set up the mapping file if they could help it. The solution we opted for was to fake up an ACL when a Windows user in a freshly converted NTFS tree viewed the security on a file. The faked up ACL would then be replaced by a real one when the user clicked OK, even if no changes where made to the displayed ACL information. Thus, over time, an NTFS qtree would migrate to NT ACLs. For phase 1, the UNIX permissions would still be stored for the file (and recomputed if the ACL changed) for NFS access. A phase 2 implementation would not require the UNIX perms since UID to SID mapping would allow NFS access to be evaluated directly against the ACL itself.

8.2 User Authentication - NIS or NT Domain?

Several UNIX/Windows file service solutions can be configured to authenticate Windows users via local /etc/passwd (or NIS) or a Windows NT domain controller. As examples, NetApp filers and Samba can be configured in this way. Many administrators would like to authenticate all users using just one authority; NIS or NT domain controllers but the password encryption issues make this more difficult than it should be. Windows NT 5 with support for Kerberos 5 will allow UNIX users to authenticate via NT KDCs. Today Samba can be used to authenticate Windows users via the *smbpasswd* file. Using *pwdump*, encrypted passwords can be downloaded from an NT domain database, stored in *smbpasswd* and used to authenticate Windows users without forwarding to an NT domain controller for verification.

One problem with using something other than an NT domain controller for Windows user authentication today is the lack of user information available when an alternative method is used. An NT domain controller will, if the requesting party supports NetLogon, return a great deal of useful information about the user being authenticated. One of the most useful pieces of information from a file level security perspective is group membership. An NT domain controller will validate the user request and return the user's domain SID, the SIDs of the domain global groups to which the user belongs, account restrictions and other useful information. Therefore today a NetApp filer has to be a member of an NT domain in order to offer the NTFS security style.

The Samba team are working to provide equivalent functionality from a Samba server.

8.3 Copying security information between servers

Support for utilities that set and copy security information is a must when transferring files during migration. *Cacls.exe*, a Windows NT utility that ships with the Windows NT 4.0 Resource Kit which allows ACL editing from within a Windows NT command shell (useful for scripting large scale security changes). *Scopy.exe* is another very useful commandline tool which allows you to copy ACL information when you copy data. Without using *scopy.exe* (i.e. if you used the *copy* command instead) the copied file would inherit the ACL of the parent directory into which it was copied. For the large scale copying of data that is usually a part of migration process *scopy* can be extremely useful.

Another migration strategy is to backup data from one server and restore onto the new server using backup tools that retain all file metadata. The NetApp integrated security design includes support for Windows NT backup utilities, allowing you to backup data from an NT server and restore it directly onto a filer, preserving all the ACLs. This is possible due to the way the filer stores NT Security Descriptors in the WAFL[5] filesystem combined with the ability to package up UNIX file metadata into Extended Attributes which the NT backup software understands.

8.4 Mapping user names

Maintenance of the username mapping file can become problematic when you are administering a group of machines rather than just one or two. One suggestion made during our customer conference was that the username mapping file should be NIS-distributable. A mechanism for centrally maintaining the username mapping file is under development and NIS distribution is one of the possibilities. We are also looking at an LDAP solution to this problem.

9. Futures

The most significant addition to the phase 1 implementation is clearly the UID to SID mapping. We are planning to use the NFS username to map to an NT SID. Unlike CIFS which is a stateful protocol, NFS' statelessness requires a mapping of username to SID for

---

[5] Write Anywhere File Layout - the Network Appliance filesystem

every request. We are designing a cache into the UID to SID mapping to address this problem. Group mapping is still an open question for reasons discussed elsewhere in this paper. Other additions to the integrated security features will be based on customer feedback of the phase 1 implementation.

10. Related Work

This section describes what we learnt from other approaches to solving the problem. We will cover Samba and AT & T Advanced Server for UNIX (AS/U) specifically, comparing our approach to two different ways of mapping NT style security onto a UNIX filesystem, one providing mapping between Windows and NFS users and the other keeping the security information separate.

Both Samba and AS/U have a somewhat different design philosophy from that of a filer in that for both these solutions the UNIX file system is the arbiter of access and metadata storage. PC file attributes may be generated on the fly (e.g. 8.3 file names in Samba) or stored in a separate file which is then linked to the object it pertains to (e.g. AS/U's /var/asu/.db).

AT & T's AS/U stores ACL information and other PC style metadata in a database (typically stored in /var/asu). Depending on the licensee's implementation of the AS/U product there may be no attempt to map file level security information between NFS and CIFS users. This means that access to a file with an ACL on it must first be checked against the ACL and then again by the UNIX permissions also on the file.

The current version of Samba does not include support for Windows NT style ACLs but more of the groundwork appears in each interim release and the ACL design is already in an advanced state. One key difference between the Samba design and that of Network Appliance is that the Samba priority has been to design the mapping between NT SIDs to UNIX UIDs and GIDs before implementing the NT ACL support. The Samba goal of replacing NT as a PDC in a domain (already working to a limited extent in current Samba pre-alpha code) means that this work has been given a higher priority that the ACL design. Current Samba Team public discussions on their ACL design seem to be focused on providing a simple but user configurable, on-the-fly mapping between UNIX permissions and NT ACLs.

11. References

1. Rob Reichel , *Inside Windows NT Security* , *Windows/DOS Developer's Journal.* April & May 1993.
2. Stephen Sutton, *Windows NT Security Guide*, ISBN 0201419696
3. Andy Watson, *Multiprotocol Data Access: NFS, CIFS, and HTTP (TR-3014)*, Network Appliance, Mountain View, California, December 1996.
4. Andrea Borr, Keith Brown, *SecureShare™: Guaranteed Multiprotocol File Locking (TR-3024).* Network Appliance, Santa Clara, California, 1997
5. Samba man pages at a local mirror. Pick from the list at http://samba.anu.edu.au/samba/

AutoInstall for NT: Complete NT Installation Over the Network

Robert Fulmer

Alex Levine

Lucent Technologies, Bell Labs

Abstract

This paper describes how we take a bare PC and turn it into a fully functioning NT desktop machine with a standard set of applications in less than 30 minutes. In 1996 we identified OS and application loading as a significant portion of our workload. Loading NT, loading our standard suite of applications, and configuring each of them to our specifications took 4 to 8 hours. This process was prone to errors that were not detected until months later. We automated the process so that an SA with our special boot disk can do the entire process in 30 minutes (60 minutes on a slow network). We do an actual NT installation. We do not clone. With our new system, a single SA can deploy up to 20 PC's per day as opposed to just 1 or 2 per day.

1. Introduction

How many of you are frustrated by the time you waste setting up NT workstations manually when there are so many other tasks demanded of you? We have found a way to automate the OS and application installation for NT so that you can spend your valuable time doing more important things than installing machines.

2. Environment

Three years ago our environment was 90% UNIX workstations and 10% PC's running Windows 3.1. Over the last two years we have migrated to Windows NT and our PC population has grown from 10% to 40%. We currently support approximately 1800 machines of which 800 are PC's. If this trend continues, PC's running NT will dominate the desktop in our area. Anyone who has supported PC's knows this is a scary trend. PC's tend to have a higher Total Cost of Ownership (TCO) than other desktops because, while they tend to be less expensive hardware, they take more work to set up and maintain.

3. The Problem

There are never enough hours in the day. We are always looking for boring, error-prone tasks that can be automated. We began to look into areas where we could save time. One of the first things we looked at was the NT installation procedure. We had a lot of new machines to install and a lot of existing machines that did not fit our standard desktop environment or needed OS upgrades from Windows 3.1 or Windows 95 to NT. We discovered that we were spending a 1/2 day to a day installing the OS and applications as well as performing our standard customizations. In addition, the procedure was tedious and error prone, as long manual procedures tend to be, so we were also spending time cleaning up mistakes and omissions after the machines were deployed. Often a mistake would not be noticed until months later when a customer used an application for the first time. This leaves the

SA dumbfounded, "How did that go unnoticed for so long?"

4. The Options

There were a few decisions we had to make when we started the project. We first had to consider the overall method of loading the box. We briefly considered disk cloning. It has the advantage of being easier to get started. You simply build a disk the way you want it. Then use that as your master and clone the disk via disk imaging software or disk cloning hardware. The cloning hardware had some limitations concerning disk size. The target disk had to be at least as big as the original and you would only get a partition as big as the one on the original disk. When we had to change packages we had the choice of doing an upgrade (not always the cleanest way) or reinstalling the master disk from scratch. We also had to worry about duplicate SID's although there are vendors out there who say they have solved this problem. (see www.sysinternals.com) This procedure only worked correctly if the hardware on the target machine was identical to the hardware on the source machine. Unfortunately we have a lot of different hardware coming in and it usually comes in a few machines at a time. Using disk cloning would have meant rebuilding the disk for each new hardware configuration that came in the door. The biggest concerns here were the type of disk controller and the motherboard architecture. We needed something that would detect hardware on the fly during the installation so that we do not need a separate disk for each hardware platform. The NT unattended installation allowed us to have this feature. We could have one distribution for all of our various types of hardware. The only requirement was that the network card be supported by the NT installation. We could make changes to the distribution without having to rebuild it from scratch each time. Adding applications and applying bug fixes was easier and cleaner. We didn't have to worry about duplicate SID's either. Unsupported network cards were not as big a problem as one would expect either. We discovered ways to add support for new network cards to the original distribution.

Once we had decided which overall method we wanted to use we needed to decide which method to use to start the installation. We considered boot proms for the network cards in our PC's, but this would have added $50-$100 to the cost of each machine and would have meant opening each machine to install the proms. Also the installation process requires several reboots and only the first boot needs to be from the net. There is no way to tell the machine "only boot from the net this time and then revert back to default next boot" as you can with Sun workstations. We also considered Linux boot disks to bootstrap the installation but lacked the knowledge to make it work. We were also hampered by the fact that Linux distributions often lagged behind on driver support for new network cards. We considered putting the distribution on CD. It would have made loads faster without impacting the network but it would have meant burning a new CD for each SA every time we fixed a bug or added new features to the distribution.

We decided to use an MSDOS Lan Manager client on a MSDOS boot disk. The Lan Manager client was small enough to fit on one floppy. The client's configuration files were plain text. And the client was not started in the config.sys file. It was started in the autoexec.bat file. This meant that we could write a program that asked the appropriate configuration questions and modified the client's configuration files. This allowed us to configure the client and then start it with the correct network card and IP settings.

5. The Solution

The manual procedure took 4-8 hours and

required that the SA baby-sit the machine and answer questions throughout the procedure. The automated procedure we designed takes at most 60 minutes (30 minutes on a fast network). The SA need only be present for the first 5-10 minutes to answer questions and the last 10 minutes to install the latest sound drivers and perform a few local customizations that we haven't automated yet.

The procedure for an Autoinstall is basically two steps. Boot from a Lan Manager floppy and answer some questions and remove the floppy once the installation has begun. The SA is then free to go off and do other tasks while the process continues. When the process is done, a few drivers are installed and the machine is ready to use.

6. How It Works, An Overview

The MS-DOS boot disk permits the SA to partition the hard disk. It then asks the SA a few questions about what Ethernet NIC is being used and the host's IP configuration (IP address, netmask, etc.). At that point, the workstation connects to an NT server and runs "part 2" of the script that is on the network drive. This program gives a menu of applications that can be installed; the default is "all". The SA chooses which applications to install. Then SA removes the floppy and leaves until the procedure is done.

This second script continues by formatting the hard disk, bringing down the appropriate distributions from the server, and running the "winnt.exe" installation program that installs the OS.

Applications are loaded last. As part of the OS installation, Data is put into the registry that causes the machine to login automatically (autologon) and run a command of our choosing by putting the command line in the RunOnce registry entry. After the OS is completely installed, control is returned to our final script. This last script installs our applications and performs local customizations. Each application has a different way to automate its installation. In general, most can be provided with an installation script file that answers the questions.

Once the process is complete, the machine is fully usable. However, with our current configuration the machine does not have the latest driver for the audio card installed.

7. The Details

The installation has four phases: the initial DOS portion of the installation, the text mode phase, the initial GUI mode phase, and the autologon or second GUI phase. The initial DOS phase is where the SA boots from the floppy and fills out all of the information that needed to install the system and then kick off the installation. The initial installation then copies all of the files needed to finish the installation and reboots. During the text mode phase the installation begins copying files to their final location and does most of its hardware detection. During the initial GUI mode phase all of the rest of the OS files are copied into place. The network card is detected and the NT networking gets started for the first time. This is the phase where the machine joins the domain. This is also the phase where the video card is detected and the drivers for it are installed. At the end of this phase the patches are applied to the OS and the settings are put in place to start the second GUI phase of the installation. During the second GUI phase all of the applications are installed and all local customizations are applied.

7.1. The Boot Floppy

There are three major pieces that are key to making this work: the boot floppy, the OS distribution, and packaging the applications.

The main idea here is to collect all of the data we need up front and then package everything in such a way that no questions need to be answered during the installation. The floppy and the initial scripts become the most important part of this procedure. The floppy needs to ask for enough information to get the machine on the net and logged into the server. The program that collects this information then rewrites the configuration files for the Lan Manager client to use, starts the client and logs into the network. Once the system is logged into the server we can run scripts or programs that ask for all of the rest of the information we need to finish the installation.

The biggest problem we had with the floppy was fitting the Lan Manager client, the programs and drivers on one floppy. We wanted support for multiple network cards on one disk but the original floppy, which was based on Windows 95's DOS, didn't have enough space to fit everything we needed. We managed to fit it all by doing three things. First, we switched from Windows 95 to an MS-DOS 6.22 boot floppy. It had smaller boot files. Second, we switched from a BASIC program (which required a 200-KB interpreter) to C++ program. Third, we started using a ramdisk and put all of the files we could in a zip file which gets uncompressed to the ramdisk. These changes gave us more space on the floppy and gave us increased performance. When we were done with these changes we had enough uncompressed files on the floppy to start the ramdisk and run the one network driver that is started from config.sys. All the initial autoexec.bat did was set a variable so we knew where the ramdisk was and then uncompress the zipfile to the ramdisk. The autoexec.bat then called a batch file on the ramdisk that ran our C++ program, started the Lan Manager client and logged on to the network. In other words, once the batch file in the ramdisk ran, we didn't even need to have the floppy in the drive anymore.

7.2. The Distribution

The second important element in the process is the OS distribution. Scripts are needed to collect whatever information wasn't collected from the floppy portion of the installation and then start the OS installation. The key to this portion of the setup is choosing the right settings in your unattended.txt file. The unattend.txt file was a file in which we could specify all of the settings for the machine that were needed to install NT without having to answer any questions. We chose certain settings that we wanted to be the same for all of our workstations and then used our initial questions to fill in the settings that we wanted to set on a per workstation setting.

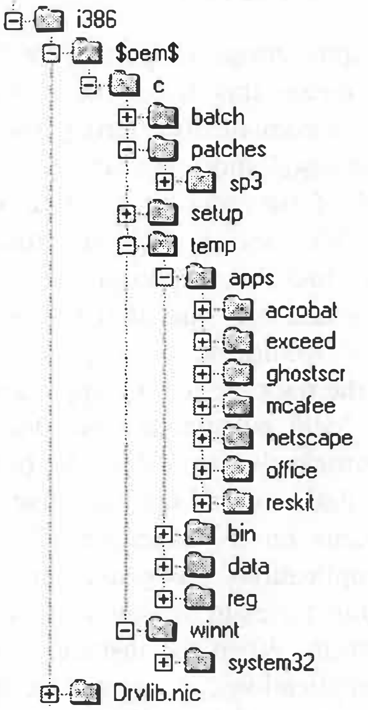
The common settings included things like installation type, CD-key, file system conversion, file system expansion, local administrator password skip, End User License Agreement (EULA) skip, and others. We wanted all of our workstations to run on NTFS so we chose to make NTFS conversion a global setting. We also chose to expand the file system to fill the disk. We chose to leave the local administrator password blank so that it would not be asked for in the middle of the installation. We set the password after the installation was done. We chose to skip the EULA for the same reason. We set the license key in the default unattended.txt because we have a site license for NT and so have one universal license key. Most people will probably want to ask for the license key as part of the initial installation. Because we are using the cmdlines.txt later in the installation and are extending the file system to fill the disk, we need to set the installation type to OEM. We also set the default display settings here. Since we wanted every machine to be part of a domain, we set the flag to join the domain. Which domain to join is set on a per machine basis since we have multiple domains. The basic idea is to create a default unattended file and put our network-wide

settings there. We then use that file as a template to create a specific file for each machine. We thought about using Universal Difference Files (UDF's). These are files that let you specify settings that override the unattened.txt file. UDF's could have been used to specify machine specific settings but some of the settings that we wanted to be able to set on a per machine basis could not be included in a UDF. We quickly discarded the idea and wrote a script that would generate a machine specific unattended.txt file from the information gathered and the template unattended.txt file.

We ran into a couple of interesting problems worth noting. One was a bug which would make the machine pause after the file system conversion if the file system was being expanded. This was done by design, but we don't know why. There is a patch on Microsoft's website that must be applied to the distribution and an extra setting that must be added to the unattended file to fix this bug. You can find this patch and information about the new setting at ftp://ftp.microsoft.com/bussys/winnt/winnt-public/fixes/usa/nt40/hotfixes-postSP2/setupdd-fix.

We also discovered that the environment that programs run in during the first GUI portion of the installation is not what one is normally accustomed to getting. We could not make any assumptions about PATH setting and other such environment settings. We also could not make any assumptions about what services were installed and running. As a result, some of the things we tried to do did not work as expected. We ended up making sure that we used full paths and keeping the cmdlines.txt file as simple as possible. All we did there was patch the OS and set up the automatic login feature and the account that we use to do the installation of all the apps and local customizations.

We tried to minimize the size of our distribution, so the only directories that we copied from the CD to our installation server were the i386 directory and the driverlib.nic directory under the i386 directory. If we had wanted to install any of the personal web server features we would have had to copy the entire i386 directory (including all of its subdirectories). We also uncompressed the files in our distribution because we found that the installation went faster if the target machine did not have to uncompress the files as well as copying them. There are also other modifications that we ended up making later that required that INF files be uncompressed so that they could be modified easily.



View of Our Current Distribution Structure

When doing an OEM installation the installation process looks for a directory called OEM in the i386 directory and copies anything in that directory to the hard disk of the target computer during the initial phase of the installation. If you create a directory called "C" under OEM then anything under that

tree will be copied to the "C" drive using the directory tree created under the OEM tree. We used this feature to copy the service pack and hotfixes over to the target machine along with batch files and executables which were not part of the original distribution that we needed during the installation. The installation process also looks for a file called cmdlines.txt in the root of the OEM directory and runs any commands found there at the end of the first GUI phase. We use this file to run the batch file that installs the patches and hotfixes. We also set up the automatic logon here and the account that we used to install all of the applications and local customizations and to set up the RunOnce registry entry that starts the application installation on the last boot.

All of our applications are part of the OEM tree which means that all of the applications get copied automatically during the initial phase of the installation. We set up a separate tree for all of the applications that we had packaged. We set it up so that each application had an appropriately named subdirectory and that one of the files in that subdirectory would be a script file which would run the package for the application and install any local customizations. One of the pieces of information we ask at the beginning of the installation was which applications the installer wants on the machine. The list of available applications was generated from the list of subdirectories in the application tree on the distribution. When the installer was done selecting applications, a master script was generated that ran the installation scripts for all of the selected applications.

We could add an application to the distribution in three simple steps. First, package the application. Second, write a script that will install it completely. Third, place it in a subdirectory with the appropriate name in the applications directory tree.

The RunOnce registry setting was the heart of the second GUI phase of the installation. Whatever command we put here was run during the final GUI phase. We wanted this to be a script that performed all application installation and local customizations. This script ended up being fairly short. We tried to package as much as we could into application packages so that when we selected to install nothing, we got a completely clean machine. We wanted this mostly for testing purposes and for packaging applications using SMS Installer. This script did two really important things. First, the script called the application installation script that we generated when the applications were selected in the beginning. Second, the script saved the resulting user profile to the machine default user profile. This assured that any new user that logged into the machine would get the settings needed to be able to run the applications that were installed on the machine.

The user profile is a directory structure plus a registry hive, a binary file which stores many of the per user application settings. The directory structure can simply be copied to the default user profile. Although the registry hive is just a file in the user profile directory, it is locked by the system and cannot be copied. Therefore, we needed to do a registry dump on the autologon user's registry key and then copy the resulting file into the default user profile. There were tools in the resource kit that allowed us to dump a registry hive. However we could only get the username from the user's environment. The registry key for the user was named by SID rather than user name. We needed to write a utility to get the SID of the currently logged on user. We wrote a wrapper program that called a resource kit tool with the right parameters and parsed the output for us.

7.3. Packaging the Applications

The third key element to this procedure was

packaging the applications so that they would not stop and ask any questions. We discovered four different ways to accomplish this: the InstallShield method, the STF file method, the vendor specific method and SMS Installer method.

7.3.1. The InstallShield Method

We found that a large majority of applications these days are using a setup program called InstallShield. This type of installation program was easy to automate. The setup routine could be supplied with the -r switch that causes it to record all of the answers given during the installation in a log file. We could then copy this log file to the application source directory and use the -s switch that specifies a silent mode installation. This mode reads all the answers to all the questions from the log file that was just created and installs the application without asking any questions. We used this method on McAfee Antivirus for NT.

7.3.2. The STF file method

We found a method that was Microsoft specific and worked on most of their big packages, such as Microsoft Office 95, Microsoft Office 97, and the resource kits. During the installation these applications generated an STF file that was designed to record the installation so that the application could be reinstalled or uninstalled partially or completely at a later date. With the saved the STF file we could give switches to the setup program telling it to reinstall everything and then specify the STF file on the command line. When we did this we found that the setup program repeated the installation that created the STF file. For example, if the STF file was c:\office.stf the command line would be setup /r /t c:\office.stf /q1. The /r switch told the setup program to reinstall everything. The /t switch allowed us to specify the path to the STF. We found that this path had to be an absolute path. The /q1 switch told the setup program to run installation in quiet mode 1. This mode displayed the progress but did not display the dialog box at the end that announced that the installation was successful. We found that the installation either moves or deletes the STF that supplied on the command line. So make sure that a copy of this file, not the original, is used during the package installation.

7.3.3. The Vendor Specific Method

Some vendors have their own setup programs or require hacks to make them work. Netscape appeared to use InstallShield but the silent mode installations did not work. We had to modify their setup.ini to change the default installation path to our standard and add other settings to prevent the installation from asking questions. In a case like this, we found that contacting the vendor was the best thing to do. In most cases they were helpful and explained how to automate their installation process. In cases where the vendor could not or would not help the final method seemed to work well.

7.3.4. SMS Installer Method

Microsoft provided a tool called SMS Installer for sites that have Systems Management Server installed. This tool watches the installation and then finds all of the files, services, registry entries, and shortcuts that are installed during the installation and scripts and packages those changes into a self-extracting archive. When this archive is run it extracts itself and performs an unattended installation of the application that was just packaged. Sometimes SMS Installer missed files and registry settings. These registry entries and files could be added to the package later through the SMS Installer's GUI. We used this to package Hummingbird's eXceed. For those sites that don't have SMS installed sysdiff is an adequate substitute for the SMS Installer. The only caveat is that the sysdiff

package cannot be modified. If sysdiff misses files or registry entries they must be added later. Missing files must be collected and copied manually. Registry entries must be put in a registry script file and incorporated into the registry manually using regedit.exe. Another tool to consider is Seagate's WinINSTALL. It's description made it sound a lot like SMS Installer. We didn't look into it because we had several solutions that were free.

7.4. Adding Support for New Hardware

One of the things that we are still working on (as we write this paper) is adding new drivers to the distribution. We found ways to add PCI network cards and SCSI drivers. We think we found a working solution for video cards but are still testing it. Microsoft did provide ways of specifying OEM hardware in the unattended.txt file but using this method would have meant asking yet another set of questions at the beginning of the installation. We chose to try to integrate the drivers into the actual distribution so that they are auto-detected during the installation. Microsoft did NOT SUPPORT this solution. They would not help us when we had problems making it work. We found that most hardware vendors tried their best to be helpful and provided some insight into what was happening during the installation's two hardware detection phases. After getting information from the vendors and poking around in the setup files, we found we were able to figure out what was going on. Once we understood the driver installation process it became much easier to find a procedure to add each type of driver to the original NT distribution. The solution we chose took some extra work but seemed to boil down to a set modifications to INF files, script files that the NT installation runs, and in some cases modifications to the initial system registry hive in the distribution.

8. Future Enhancements

We have plans for several future enhancements to AutoInstall. We are working on a way to install drivers that the NT installation does not auto-detect. We think we may be able to use SMS Installer to create a package for each of them so that they can be installed automatically. We are planning on moving the applications tree out of the OEM directory tree so they can be copied selectively. This way only the applications chosen by the installer will be copied. We are also planning on adding DHCP support so that our installers don't have to answer as many questions as long as a DHCP server has settings for the client computer.

9. Suggestions for Vendors

There are many things that vendors could do to make this automation process easier. A set of tools to add new drivers to the NT distribution would be nice. A standard method of packaging applications would also be a great help. It would also be really nice if applications were less intrusive in the operating system. Having an application install extra files in the system directories does not appeal to us. If the applications were less intrusive, it would be easier to centralize them. NT workstations could become more generic and easier to setup and to maintain. Another nice feature would be a "smart" application which knows what settings it needs to run and adds reasonable defaults to the user's profile and the local machine if they don't exist. They could at least give a reasonable error message upon startup, stating which settings are missing on the current machine so the user can fix them or call an SA if necessary.

10. Findings

We have a similar system for our Sun Solaris (UNIX) hosts so we knew what benefits to expect. As we anticipated, the result was a fundamental change in the way we install and

upgrade hardware on our network. Previously SA's spent most of their day doing installations. Now that has become a minor part of their job. They spend their time focused on other tasks such as attending to customer requests and trouble-shooting.

We had expected that it would be useful to have every detail of every machine be configured the same way. However, we underestimated how useful AutoInstall would be to our front-line support personnel. Our front-line support had been spending a large amount of their time with issues from customers with newly installed machines that could be traced to human error during the manual installation process. This automated installation process nearly eliminated these problems. As a result, our front-line support personnel now can focus on deeper problems. This was a benefit we had not seen in our Sun Solaris (UNIX) environment. On those systems, applications could be centrally installed, unlike NT where applications and their configurations are loaded locally.

Our customers are extremely satisfied with the accuracy of our installations and our ability to roll out machines in higher volume.

11. Conclusions

Automating OS and application installation was difficult to set up initially because we were breaking new ground. Now that the process is in place, it should become more commonplace. The result of this automation has been a productivity boost on three levels.

(1) Our installers can install more machines per week error free. In fact, for the first time, they can install more than one or two machines per day.

(2) Our front-line support personnel no longer receive problem reports from customers with machines that were misconfigured due to human error.

(3) Our customers are happier with their environment because machines are more likely to be installed on time and the machine they receive is configured correctly.

Obviously, the happiness of our customers is the most important gain.

12. Availability of Source Code, Etc.

Contact the author's for code availability.

Robert Fulmer rjf@research.bell-labs.com

Alex Levine alyank@research.bell-labs.com

A Comparison of Large-Scale Software Installation Methods on NT and UNIX

Michail Gomberg, Rémy Evard, & Craig Stacey

Mathematics and Computer Science Division, Argonne National Laboratory

Abstract

Our computing environment consists of hundreds of UNIX and NT-based computers. We have a coherent UNIX software installation model that scales comfortably to hundreds of machines. We have spent a great deal of time in the last year learning to understand the software installation and support mechanisms for a similarly large NT infrastructure. In this paper, we examine the underlying requirements for large-scale software installation support, and compare and contrast the NT and UNIX environments, identifying strong points, weak points, and issues.

1. Introduction

The computing environment in the Mathematics and Computer Science Division of Argonne National Laboratory consists of nearly a thousand computers, including supercomputers, servers, workstations, desktop machines, and laptops. The majority of these are running UNIX, but the number of NT machines is steadily growing, particularly on desktops.

This set of machines is managed by a group of seven system administrators, meaning that, like most systems administration groups, we have more than enough work to go around. Our ability to administer the environment relies on scalable techniques. We have to manage large sets of machine as if they were one single system, most preferably from one single location. We were originally nervous about our ability to do this in a growing Windows NT environment due to its reputation as a system requiring hands-on administration. Software installation was one such area that we were worried about.

On each of our UNIX architectures, including SunOS, Solaris, Linux, FreeBSD, AIX, and IRIX installations, we use a common method of installing software. It scales well, in that we are able to install a given piece of software once and have it automatically work on every machine of a given type. This system is described in detail below.

As we began to look at software installation on NT in detail, we began to feel that our fears were justified. The usual software installation methods on NT require the administrator to sit in front of an individual machine, answer questions interactively, wait for the software to load, possibly reboot the machine, and then do a lot of tweaking in order to make the software available to other people who might log in to the machine. This approach doesn't scale to hundreds of machines. Fortunately, there are ways around most of these problems, but, as described below, they are not uniformly applicable to all NT applications. As a result, we started a long-term project to understand NT application installation better, to test the various installation tools, and to develop a philosophy and method of software installation that scaled as well as the methods we use on our UNIX machines. Ideally, we would learn some new approaches with NT that we could then apply to our UNIX systems to help some rising problems on those systems.

In this paper, we describe the software installation methods that we use on our UNIX machines, because they provide a context for our desired result. We explain what we have learned about NT applications and why we feel they should be considered differently than typical UNIX applications. Finally, we describe the system that we have begun to use, explain how it helps us, and identify remaining problems.

This is an interesting time for NT administrators because the range of tools is changing so quickly, due primarily to Microsoft's growing effort to help solve scalable administration problems. Many of the tools that we are now using didn't exist when we started this project, and most likely, some tools will be released between the time this paper was written and the time it will be presented. However, we believe that our desired solution, our observations on applications, and our current approach will still be of use to other administrators who have to surmount similar problems.

2. UNIX Software Installation

When we started our NT application installation project, we initially looked at our existing infrastructure to identify the components that we felt worked well for us. These helped shape our concept of an ideal NT solution. We present a brief overview of our UNIX software installation here.

All of our UNIX machines are interconnected over our internal LAN. We manage several different flavors of UNIX, but we have the same software installation methods for each of them, so we will only discuss the Solaris implementation in detail. Although we don't use Depot [1] because it doesn't solve all of our problems, any readers familiar with that software installation method will recognize the concepts presented here.

The software installed on our Solaris machines can be broken into two different categories:

1. Software that is installed on the local machine. This includes the operating system, some daemons, and any licensed third party software that has to run on a specific system.
2. Software that is installed on a networked file system. This includes the vast majority of software installed on Solaris, including programs that have graphical user interfaces, application suites, and simple command line utilities.

Locally installed software is usually installed at build time by our Solaris build scripts. If we decide to install a new piece of software locally onto a machine, then we install it on every Solaris machine. This is very important - it is much easier to manage all of our Solaris computers as a single unit if each of them has an identical set of installed software. We modify the build scripts to ensure that new Solaris machines also get this software. As mentioned above, this is a rare occurrence.

Most of the installed software on our Solaris machines is installed into a networked file system. In our case, the path to this networked file system is /software/solaris, but this same concept is more commonly available as /usr/local at other sites. Each Solaris computer mounts this single network repository. This means that, once a piece of software is installed in /software/solaris, every Solaris box immediately has access to it. In the overwhelming majority of cases, no extra work is required on any computer in order to make the software available.

The /software/solaris filesystem is exported via NFS. It doesn't deliver software quite as quickly as local disk would, but that has never been an issue. It will scale reasonably to several hundred client machines. Beyond that point, we will replicate the directory and serve it from multiple servers, which will require a minimal amount of work.

One serious problem with this approach is that if the network server goes down, then all of the Solaris hosts are crippled. In our environment of relatively reliable networks and computers and a relatively small number of administrators, we deem this to be an acceptable tradeoff. Further, we replicate the /software/solaris file system to a disk on a second server, and have the clients automatically fail over to the second disk when possible. This helps when the server has a problem, but doesn't solve the case of a network outage. In practice, this has not been an issue. (If the network is down, there are usually bigger problems.)

When a piece of software is installed, we put it in its own directory under /software/solaris. The path is hardcoded to include the software's category and version number. For example, emacs is in /software/solaris/apps/packages/emacs-19.34.6/. All emacs files of this version are stored under this directory. This allows us to install many different versions of emacs without any collisions between them. This is crucial.

Not every package can be easily convinced to live under one directory, but in practice, we have always been able to get around this. UNIX software is typically easy to modify with configurable files, and in the most cases, can be fooled with symbolic links.

All that the user needs to do to access most applications is to put this directory in their PATH environment variable: /software/solaris/apps/bin. We populate this directory with symbolic links to the actual executables that live in their own directories, and, using the "soft" [2] system, we modify the user's startup shells to update any environment variables necessary. We can change the default

version of an application with one quick command. Again, we trade off a minor performance hit (symlink resolution) for increased flexibility of administration.

Our UNIX installation is not without problems. As the size and complexity of applications grows larger, and the cost of local disk grows cheaper, we find that we would like to install more applications on individual machines. Doing so is more problematic than installing them centrally. Also, an increasing number of third-party applications have arbitrary restrictions that nearly force us to do by-hand installations on individual machines. Our general feeling is that in these cases, those vendors are borrowing the worst ideas from the world of PC applications.

The key points of our UNIX installation methods are:

- Each machine is identical to every other machine of the same type.
- Most software is installed in a network file system, and in this case, no extra work is required to make software available on all machines.
- All information related to an application version is kept in one place -- the directory for that software.
- No extra work is required to make software available to all users.
- We make heavy use of symbolic links to glue different areas of the file system together into a seamless interface for users.

3. NT Software Installation

One of the hardest tasks that we have with our NT environment is installing software. Some people have expressed surprise at this. After all, they point out, it's much easier to install an application under NT. All you have to do is put the CD in the drive, click setup, and answer some questions as InstallShield goes to work. One would expect that UNIX software management would be much harder, since there is no uniform way of installing applications, configuring them requires editing various arcane files, and sometimes they even have to be compiled. This is true; if one's goal is to build a single machine for a single user and install a given set of applications, it is much easier to do on NT than on UNIX.

However, managing a large environment for many different users is entirely different from building an isolated machine. Ironically, it is the NT application's focus on the user and on the single machine that primarily makes it so much harder to maintain on a large scale. But this is not the only difference between software on NT and software on UNIX, and not the only reason that large-scale NT software installation is difficult.

In this section, we identify the issues that have made NT application installation challenging in our environment. It is our hope that some of these issues will be addressed in future releases of NT and the various applications. Understanding these issues (and why we consider them to be issues) is also key to understanding how we install applications.

3.1. NT Software is GUI-based

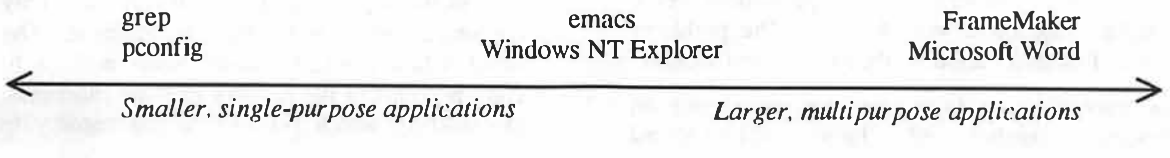
The first difference noted by most people is that NT's applications generally have graphical user interfaces, while UNIX applications may or may not. We find that whether or not an application as a GUI really has no bearing on how difficult it is to install.

A somewhat related issue is whether or not administration tools should be based around GUI. Generally, GUI-based tools are harder to automate and extend. Large environments all have their own particular requirements, so administration tools for them require customization. At the present time, command line tools are easier for us to use when managing a large, diverse environment.

3.2. NT Software is More Complex

Most of the applications that we install on NT have a different focus than those we install on UNIX. To illustrate this point, see the line in Figure One. On the far left-hand side of the line are small, single-

Figure One: Software Continuum



purpose command line utilities such as "cat", "grep", and "traceroute". On the far right are large, multipurpose monolithic applications such as "Office 97" and "Internet Explorer". These applications tend to provide a very large set of features from a single interface. Applications like emacs and perl lie somewhere in between.

The majority of the applications that we install on UNIX are on the left side of the line, while the majority of the applications that we install on Windows NT are on the right side of the line. This is not to say that UNIX has no large, monolithic productivity-based applications. It certainly does. But in our environment, UNIX owns the left side of the line, while Windows NT owns the right side. We believe that the wealth of these types of applications for NT is the primary reason for its steady rise in popularity.

The focus of the program generally does not have an impact on installation. However, the applications on the left side of the line are often easier to install, regardless of their OS. There are several reasons for this.

- Footprint. The smaller the application, the smaller the fixed storage footprint. When applications are stored on network file servers, the application load time becomes an important factor. For small programs this is often not an issue, but for very large ones, the startup delay can be painful. As a result, one leans toward installing the application on a local disk rather than on a network file server, making it much more difficult to install and update.
- Options. Typically, the larger and more complicated an application, the more options it has at install time. And, during the lifetime of the application, the larger, more comprehensive applications seem to want more add-ons and more components. This makes it quite a bit more difficult to get a single installation that will satisfy a large group of users.
- OS integration. Large applications are very nearly operating systems in and of themselves. (Emacs used to be accused of this, but it has been far outclassed by applications in the NT world.) They depend on a large portion of the OS, and they often have a very strong reliance on the OS being set up in an exact manner. On NT, this has resulted in the applications being tightly integrated with the OS. The problems with this under are described in the next section.

It is interesting to note that we are seeing an increasing number of large, GUI-focused applications available for UNIX. This is one of the areas where we hope that our lessons learned from NT application installation will help us out on the UNIX side.

Also, there are more utility-style programs available for NT, including ports of perl, various UNIX utilities, and a batch of new, NT-specific tools. Unfortunately, it is still difficult to install networked versions of these tools to be shared by all the workstations from a single location, usually because they are too tightly bound to a specific machine.

3.3. NT Software Lives in a Single-User World

Historically, NT applications come from an environment where they could make certain assumptions that are no longer valid:

- The application would only be installed on a single computer.
- Only one user used the computer.
- The application would be installed on local disk.
- The application might have to add things to the operating system.

This set of assumptions, combined with the usual large size and complexity of NT applications, has resulted in a set of conventions for NT application configuration that make large-scale installation particularly difficult.

- NT applications are too intertwined with the OS. Most applications depend on mfc42.dll or shell32.dll to be present on the machine. When these files aren't there or are different from the expected version, software will replace them with the version that they were built with. This can cause other applications to crash or act oddly. Since they also often copy in other DLLs, they tend to bloat system32, making it more difficult to install network versions of the application, and also rather difficult to decide what is safe to uninstall. Also, applications that are written for Windows95 will often try to write to areas that are protected under NT and require Administrator privileges.
- NT applications are too machine-specific. Most applications write their configuration information into the registry. Since the registry is machine-specific, the software is only configured correctly for that one machine. The usual solution, which doesn't scale well, is to copy portions of the registry to other machines. Discovering which portions of the registry to

copy, and doing so without causing side effects, is a challenge. Some software also makes use of a user's environment variables, which, unlike UNIX, are set up on a per machine basis, as part of the local registry.

- NT applications are user-specific. This shows up in two ways. First, when an application is installed, it will often add itself to the current user's Start menu. Other users, if they login, will not be able to run the application, even though it's installed on the machine. This can be fixed by making sure every application is added to the 'All Users' portion of the Start menu. Not all applications are smart enough to do this yet. Second, applications tend to store user-specific data in various places on the local machine, including the registry, user directories, application directories, and even operating system directories. So, for example, one user may login to a particular machine and be able to read some other user's email because the email program wasn't programmed to handle multiple users.
- Roaming profiles aren't quite right. Roaming profiles attempt to solve some of the above problems, but they create new issues. When applications are installed on a single machine, roaming profiles end up pointing to applications that aren't there. If a roaming profile is loaded onto a workstation that has a differing installation of the same app, they munge the correct settings.

Because of these differences, we have to approach NT software support completely differently than UNIX software support. Overcoming the above problems requires special tools, special care, and a lot of planning.

4. Possible Installation Approaches

In order to decide on which installation methods we'd ultimately use, we had to evaluate and consider as many options as we could. Some of these have been used by other sites, some of these we came up with on our own. Our final solution was a mix of these.

4.1. Installation By Hand

The first option is the obvious: by hand. You sit at the machine, pop in the CD-ROM, answer all the questions, wait while it copies files, you answer some more questions, and eventually you're finished. For a minor optimization of this technique, you can put copies of the CDs or floppies on the network, and install from that.

The by-hand method is usually done by sitting at the console of the machine, but it's also possible to bring up a remote console of the machine using various third party software, including the SMS Help Desk application. Neither of these is what you really want to do, as the by-hand approach almost always involves too much of time overhead. Worse, for each iteration of the install, you introduce the chance for things to change from machine to machine, such as accidentally selecting different options.

We decided that by-hand installs should be avoided if at all possible. (However, if you have a lot of free labor lying around, this may be the easiest solution.)

4.2. Installation at Build Time

For a default set of software that you know will seldom change, arranging for the software to be automatically installed at build time is ideal. This method is automated, it scales, and it remains consistent for each machine. After an automated build with software install, you have a base machine with a known set of software. You know exactly how that machine is configured.

In order to automatically install software at build time, you usually run a command script after the initial OS install, or initiate an automated administrator login that executes a series of install scripts. In our current solution, we ended up doing both.

The first option, running a command script, we use to apply the latest NT service pack and hot-fixes -- admittedly more of an OS issue than a software issue. We could make more use of this, however the system needs to reboot after the application of the service pack, so we turned to the second option.

To use an automated login, it's best to create an account dedicated to that purpose. That way you can disable it when you're not using it, but, more important to the task, you can assign it a specific login script. The script should execute any post-install cleanup that couldn't be done before the first reboot, and then launch an installation script of your choosing, be it a batch file, a perl script, a CMD file, or what have you. Properly done, this can spawn the appropriate package installs, one at a time, and in the order you specify. The order can be important -- for example, getting perl or Winzip on the system first would be prudent as later installs will want to use them.

We experimented with a third option: using the OEM install tools found in the Resource Kit [3], notably sysdiff. With sysdiff, you take a snapshot of a machine before an installation and a snapshot after. Sysdiff records the differences, which, presumably, can then be applied to any other machine to install that same piece of software. We found, however, that sysdiff did not scale well. We started using it when we were first mass-building our NT environment. As is common in these cases, what we thought should be a default build at the start of the project and what we eventually discovered should have been the default build were vastly different, the latter being a much larger set of applications. With each attempt to add an application to the sysdiff package, it took more and more tweaking to the various configuration files to make the creation of the snapshots actually succeed. Sysdiff also had serious problems when our machines weren't compatible at a hardware level. More often than not, as the default build grew, sysdiff would fail miserably, usually with a fatal error at the last step of the procedure, creating the distribution directory.

However, installing software at build time has one serious problem. It only fixes the installation problem once, which is fine if you never expect to install a new application in the future. However, if you do upgrade your software, you have to modify the software that is installed at boot time. This changes the known base, so you are likely to end up with inconsistent machines. Worse, you probably will also have to install this application on machines that were built months before. This essentially puts you back where you started -- trying to figure out how to install software on existing machines.

Installing software at build time, then, is not a complete solution, but it's certainly useful to be able to do it.

4.3. Automated Remote Installation

What you really want is some way of automatically installing software on remote machines. We considered several different methods.

SMS has other uses besides remote control. It also has a feature called the "package command manager". This allows the administrator to specify software 'packages' that can be installed on workstations in one of two ways. She can schedule compulsory software packages that will be installed at a certain time. For optional software, the user can be presented with a list of software that can be installed at login. The second option helps with the problem of buying licenses that you don't need, allocating expensive software packages only to those who need them.

In order to use the package command manager, you need to bundle your software into a package. Creating a package involves finding or creating a single-command install option that is usually silent to the user, defaults to all the right options and preferably runs in the background. For some applications, this is trivial. For others, it can be quite painful, but we've yet to find a package we absolutely can't do this with. As a last ditch resort, we wrap the setup program in a Visual Basic program that basically clicks the correct buttons automatically. (This borders on ridiculous, but it works.)

It's also possible to do remote installs without SMS. The administrator wraps the software installation in the same kind of package, and then uses rsh or rlogin to connect to the machine and execute the setup command locally. The Microsoft supplied rsh is less than ideal, so we use the Ataman package. The advantage of using rsh is that it runs when you tell it to, in contrast to Package Command Manager, which has its own internal scheduling mechanisms. If you need to push a hotfix out right away, you will want to use rsh.

An administrator can also use the NT scheduler service to execute commands on remote machines. This is again similar to the package command manager method of installation. We only briefly toyed with this idea, as it involves configuring the machine to run the scheduler service as a user with administrative privileges, and we were unsure of the security aspects involved. The other options seemed to provide the same functionality, but this method is an option to consider.

In essence, remote installation consists of three issues:

- Wrapping the software in some kind of package that essentially turns the entire install into one single command.
- Initiating that command on a remote machine. SMS handles this the most gracefully.
- Worrying about the permissions with which the remote command will execute.

4.4. Installation Via Replication

Another approach is to replicate a disk image onto the local disk. (We've heard of some sites that boot into linux in order to update Windows95 FAT partitions remotely.) The replication can come from a CD or a networked disk. The problem with this

method is that it essentially involves constantly erasing the local disk, which has two side effects.

First, this prevents any user data being stored locally. This may not be so bad if your users are careful about keeping their data on a network server, and if you store profiles on the servers. But we've had interesting problems with that solution, particularly in an environment that has a lot of laptops. In terms of a time investment, this can be almost as bad, or perhaps even worse, than manual software installations.

Second, there's an inherent SID problem replication brings up, in that that the machine's SID needs to be retained, or in the case of building new machines, a new SID has to be created. Finally, not all machines in an environment share an identical hardware base.

The replication solution just doesn't scale to a large environment, at least not with today's available replication software.

4.5. Remote Booting

Finally, one can consider remote booting by using a boot PROM on the network card on a machine with no local disk. From an administrative standpoint, this is a dream setup. From a user standpoint, this is the worst possible solution, as you've sacrificed local disk performance for ease of installation. If this is the option you go with, you should probably consider some of the Windows terminals that are available, or wait for the NT 5 caching schemes.

5. Requirements for a Software Installation Solution

When we first started installing NT machines in our workplace, we took the simple solution to installing software: we installed it by hand. That worked fine as long as there were only a few machines, and those of us using them knew exactly what we were doing. But the situation quickly got out of hand, and we realized were spending a lot of time walking from one machine to the next, a lousy solution that we've never had to implement on our UNIX machines.

At that point, we went through our analysis described above - understanding how NT software is installed, considering various options for large-scale installation, and identifying what we liked about our solution for UNIX.

We decided that whatever solution we developed had to have the following requirements.

- The installed applications need to achieve reasonable performance. Our first attempt was to use a network installation for major software. Performance was miserable. Not only did users complain, but in some cases, the software crashed. This was in part due to slow network storage, and in part because the applications didn't handle non-local disk installation very well. So, this meant that we had to plan to install most software onto local disk.
- The distribution to each machine had to be automated. We refuse to do more walking from machine to machine. We'd like to visit each computer only twice in its lifetime: once to install it, and once to take it away. We should be able to do everything else remotely. Note that this isn't because we're lazy, it's because visiting each computer simply doesn't scale.
- Our solution should attempt to keep the application from interfering with the operating system. This would help reduce mysterious problems with the operating system and collisions with other applications.
- We would like every machine that we manage to be as identical as possible, differing only when the hardware configuring forces it. The operating system, the patch levels, and the applications should otherwise be the same. This allows us to generalize across the machines, to simply know, without looking, that an application is installed or that some command should work. Unfortunately, this doesn't work in the NT world for two reasons. First, no machine has that much local disk space. Second, the licensing cost would overwhelm us.
- More realistically, we decided to define several classes of machines: developer machines, visualization machines, student machines, secretary machines, and so on. Computers in a particular group all have the same software installed on them. This is not quite as good as having one flavor of machine, but it still lets us generalize.
- We've learned not to trust applications that have been known to mess with the OS. We'd like to have a way of discovering what they did at installation, and what they did over time. Ideally we would have a way of checking each machine regularly. This would help us maintain our goal of keeping every machine the same.
- Our solution should be as simple as possible. (It's currently not.) For example, it should be

possible to build a machine and all the software on it by popping a floppy in it, booting, and walking away. Hours later, it should have all software installed on it correctly. Afterwards, when we update software across the net, it should only take one command, and it should automatically happen on all of the appropriate machines.

- We have a lot of students who work with us for short periods of time. They should be able to come up to speed on the tools that we're using relatively quickly. This means that an encyclopedic knowledge of NT and NT tools should not be required.
- Our method needed to help us keep track of installed software so that we could ensure that we are legal with respect to licenses for installed software.
- Finally, we're aware that the NT world changes pretty quickly. We didn't want to spend a lot of time building a custom solution that would be useless in a year. Instead, we wanted to build it out of existing tools, using Microsoft-sanctioned methods, so that we would be more likely to be able to adapt to changes.

6. Our NT Software Installation Approach

There are three main components to our current NT software installation method.

1. A repository of packaged software.
2. A build procedure that installs a pre-determined set of packaged software on new machines.
3. An update process that installs packaged software on existing machines.

When we have a new application to install, we package it, put it in the repository, and arrange for new machines to install it. Then we initiate a network push that installs that software on all existing machines. This solution doesn't quite meet all of our ideals (for example, packaging software is not particularly simple), but it's a rational method that has reduced the amount of legwork and helped to keep our machines running consistent sets of software.

6.1. The Software Repository

The software repository is a networked share where we store software that is ready to be installed. Each of these applications is "packaged", meaning that it can be completely installed by running a single command.

The majority of the work here comes in creating the package. In many cases, applications already have a silent or unattended install option, and the package simply consists of running 'setup' and directing it to the correct "answer" file for its defaults.

In other cases we have to use additional software to prepare the package for unattended install. Internet Explorer and Microsoft Office, both required the use of their resource kits to create an unattended distribution.

And then there are the applications that have no concept of unattended install. For these, we write simple VB apps which send the correct key sequences to the installer applets.

6.2. The Build Procedure

We use the NT 4.0 unattended installation procedure and our own boot floppy to initiate the install process. We found that we can significantly reduce the initial build time by following the Microsoft recommendation to remove portions of the NT distribution from the distribution location on the network. The initial build time, which includes formatting a portion of the workstation disk, takes about 25 minutes per machine on a fast ethernet network. The initial distribution applies the latest service packs and hot-fixes, installing them by using the "run at install time" feature in unattended install mode.

Once the OS is installed, our build procedure adds a registry key for auto-logon of a predetermined account at boot. The machine reboots, and logs in as that account. The logon script for that account registers the machine with the SMS database. Next, the script initiates the step that installs all of the packaged software. First it looks to see what class of machines this computer is (for example, secretary or developer), from which it can determine which applications need to be installed. This length of time that this takes depends on the number of applications that we add, but 10 to 15 minutes is about average.

Finally, the login script makes some local registry modifications, copies in some shortcuts, and adds printers.

Setting the network correctly is tricky, especially since the machine is usually moved from our lab to someone's desktop after it has been built, which may require that it get a new network address. When the machine is initially built, we use DHCP to let assign an IP address from the server's pool of available

addresses. This is how DHCP servers are normally used. However, we don't use Dynamic DNS, and we want DNS to work for every machine, so we have to make sure that once a machine is built, it gets the same IP address from that point on. We accomplish this by configuring the DHCP server to reserve a specific IP address for a client MAC address. We obtain the MAC address from SMS during the part of the build process in which the new machine adds itself to the SMS database.

6.3. The Update Process

Once a piece of software is packaged, installing it on a remote machine is simply a matter of remotely invoking the package with the right permissions. A number of ways to do that are discussed in Section 4.3.

For the most part, we use SMS to push software packages out, because it runs with the right permissions and it gracefully handles machines that are temporarily off. Occasionally, if we're in a hurry, we use rsh rather than SMS. At the moment, these pushes are initiated by hand, but they could be automated to ensure that every machine of a certain class always has the correct software.

6.4. Observations on Our Approach

The approach we've taken is working very well for us. We integrated the best parts of what we researched, and came up with a reasonably automated system that requires little administrator presence at the machine after the initial build.

Being able to specify that all machines get the latest hot-fix without having to leave our office is a big win from an administrative standpoint. In fact, that very situation came up at our site after the recent denial of service attacks aimed at .gov sites.

We are able to keep a steady base level of machines by ensuring that when a software package is added to the default build for new machines, it also gets installed on the existing base of machines. This keeps guesswork at a minimum when it comes to troubleshooting or license tracking. It's about as close as we can come to a unix environment, where all applications sit in a common nfs-shared repository. We at least can be assured of a common platform.

Using these methods, the only differences between machines within the same group (general workstation, development, etc) are at a hardware and driver level. It's still not the perfect world, but it's at least as level a playing field as unix.

We have a few problems left. We don't have a way of carefully monitoring all machines to see what has changed during an installation, and we occasionally run into problems with this. While we think our current solution is pretty simple, packaging a piece of software can sometimes be fairly tricky. Our biggest problems, though, are associated with the basic NT installation issues that we can't fix: the tendency of an application to be too closely tied to a specific machine and user.

7. Comparing NT and UNIX Installation Methodology

Having implemented fairly comprehensive software installation methods for both NT and UNIX, we find that neither of our solutions is completely satisfactory. Indeed, there are features of our UNIX machines that we wish we could use on NT, and aspects of the NT installs that would be very handy on UNIX.

7.1. Our UNIX Wishlist

Here are the features of our NT application installation that we wish we could easily duplicate on UNIX:

- Better fault tolerance. When the UNIX file server with all the applications crashes, which is rare, everyone is in trouble. On NT machines, nearly all of the applications are local, so people can continue to work even when the network crashes. One way to fix this with UNIX would be to have smarter automounters that can handle server failover. Another would be to install applications locally on each machine, just as we do now with NT.
- A common push/pull method that's better than rdist.
- Even fewer machine-specific applications. We're worried about the trend we see in some UNIX software that forces us to install applications locally and then node-locks the licenses.
- A one-step configure, build, and install mechanism similar to NT's InstallShield.

7.2. Our NT Wishlist

- Symbolic links.
- A clean, enforced separation of the operating system, applications, and user space.
- A clean separation of multiple versions of the same application. It's very handy to be able to

try out a beta version of an application without having to move completely away from the stable version.

- More useful profiles. Profiles go a long way towards capturing a user's configuration, but not enough applications use them, and roaming profiles just don't work right. Storing profiles on the server, with options to copy, link or create machine-specific profiles, similar to the irix method, would be a better solution.
- Less reliance on the local registry. Too many applications use it for storing configuration and state information. This doesn't allow for a consistent work environment for users that use more than one machine. We'd like to see software installs stay away from the HKLM registry tree altogether, and instead of editing HKCU, putting configuration settings into a globally available repository.
- Consistent silent installation mechanisms. There are too many different installer technologies for Windows, each with its own learning curve and methodology, and many of these installers don't support a silent install option, forcing ugly hacks.

8. Conclusions

We set out to develop a scalable software installation procedure for NT that was as useful to us as our existing UNIX installation strategy. After learning about NT application issues, many tools for NT installation, and trying out various options, we have developed a solution that, while unlike UNIX strategy, is still fairly scalable. During the process, we identified some features of NT that we would like to replicate on UNIX, and vice-versa.

Our remaining difficulties with large-scale support of NT application installation are intrinsic to the way in which NT applications interact with the operating system. We know that some of these issues will be addressed in future versions of NT, and look forward to that event.

9. Author and Project Information

Michail Gomberg is a systems administrator in the Mathematics and Computer Science Division at Argonne National Laboratory. He was the lead technical architect for this project. His email address is gomberg@mcs.anl.gov.

Rémy Evard is the manager of Advanced Computing Technologies and Networks in the Mathematics and Computer Science Division at Argonne National Laboratory. He is actively pursuing research in systems administration, with the hope of making it less difficult and more fun. His email address is evard@mcs.anl.gov.

Craig Stacey is a systems administrator in the Mathematics and Computer Science Division at Argonne National Laboratory. He was the one whose soles were saved with this procedure. His email address is stace@mcs.anl.gov.

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

10. References

[1] Colyer, Wallace & Wong, Walter, *Depot: A Tool For Managing Software Environments*, LISA VI Conference Proceedings, 1992.

[2] Evard, Rémy and Leslie, Robert, *Soft: A Software Environment Abstraction Mechanism*, LISA VIII Conference Proceedings, 1994.

[3] Microsoft Windows NT Workstation 4.0 Resource Kit, Microsoft Corporation, Microsoft Press, 1996.

Appendix A: Tools

The following is a list of many of the tools that we've found to be very useful.

NT Resource Kit:

Srvinfo	A very useful utility to look at services on a remote machine from the command line.
Shortcut	Allows us to copy .lnk files to the right places without remaining linked to the source location.
Windiff	A graphical comparison tool that can be used to look directory differences.
Instsrv	Allows installation of services from the command line.
rkillsrv	Allows processes to be killed remotely.
rconsole	Remote console for NT.
setupmgr	GUI used to create a base answer file for the unattended setup. We had to do a lot of tuning to this file to get it to be really useful.

NT Rollout tools:

Sysdiff	Creates snapshots and difference databases of a machine after software is installed.
IEAK	Internet Explorer administrator kit. It creates a custom distribution, which can then be installed in unattended mode.
ORK	Office resource kit. Same function as the IEAK, but for MS Office.
Winnt	This is the NT installer that comes on the NT 4.0 CD. We use it to install NT in unattended mode.
regedit	Merges registry edits into the machine registry.

NT Services:

Dfs	Lets us create a single distribution location, spread over multiple disks and multiple servers.
SMS	SMS provides a database of clients on the network, along with the ability to schedule commands for execution across the clients. Also includes software auditing and remote control.
SMS Package Command Manager	This service comes with BackOffice resource kit and allows SMS jobs to run unattended.
DHCP Server	We use DHCP to get the machine onto the network initially, without having to assign an IP address at build time. Once the machine is built, we use DHCP to assign a specific IP address to it, by getting the MAC address from SMS.
Scheduler Service	Can be used to run jobs remotely, however there are security issues regarding 'system' account.

Other Microsoft Tools:

DOS 6.22	Boot disks need to be DOS 6.22 in order to fit all required files on it. Windows95 or 98 system files take up too much room.
MSLANMAN	We use LAN Manager for DOS to copy the NT files onto the workstation disk.
MSDN CDs	Incredibly useful for getting the latest information, especially pertaining to the reduced TCO initiatives from Microsoft.

Windows Development Tools:

Visual Basic	We use VB to run queries on the SMS database, and create automated installations for applications that don't have an unattended install option.

Third party:

Ataman rsh	Execute commands remotely on machines.
Perl5	We use the Activeware version. Without this we would be sunk, or at least very unhappy.

Software Distribution to PC Clients in an Enterprise Network

Cameron D. Luerkens, H. John Cole, Danielle R. Legg
Rockwell Collins, Inc.

Abstract

Rockwell CACD and CCA divisions merged into Rockwell Avionics & Communications in March, 1997 (now Rockwell Collins), creating support issues for integrating and maintaining several networks that included Banyan, Novell, Microsoft NT and PC-NFS (Hummingbird's Maestro). The combined total PC clients now number about 10,000, which include major sites in Cedar Rapids, Iowa; Dallas, Texas; and Melbourne, Florida. Other Service Centers and off-sites are also included in the total.

Introduction

A Common Network/Desktop planning team was formed in May 1997, to make plans to implement a **Common Network Operating System** (CNOS) and a **Common Desktop**, to be implemented across the 10,000 PC clients by September 30, 1998. The plan included transitioning the above sites, some with their own domains, to one NT Domain (CCANET). This included a mix of UNIX and NT servers providing Authentication, File and Print services, and Security. The plan also included delivering standard applications locally to the PC clients. As of June 1st, 8,000 of the 10,000 clients running Windows 95/NT have been converted to CCANET domain with the CNOS standard in place. In addition, installing the standard applications locally has completed the Standard Desktop requirements.

The CNOS implementation utilizes UNIX and NT servers in the following ways. UNIX home directories are being used for all PC clients. Global applications and data storage are accessed using both NT and UNIX servers. Authentication, printing and Systems Management Server (SMS) services are administrated via NT servers. This strategy allows for maximizing the utilization of current UNIX and NT resources within the company, thereby reducing overall costs. A third party product, Syntax's TAS, provided the UNIX process to allow transparent mounting of UNIX drives to a PC in the NT environment. The client implementation provides delivery of over 80 common network applications with access via the Start Menu. Also, four highly used applications are loaded locally.

The following discussion centers on the process of delivering the first four applications locally to the 8,000 clients, ensuring a common desktop. The applications are Microsoft's Office 97 Professional, Command Software's F-PROT v3.0, Netscape Communications' Netscape Navigator v3.01, and Adobe's Acrobat Reader v3.0. The distribution of these local applications was done through the use of several tools and methods due to process requirements, unique software configurations and bandwidth limitations.

Strategy

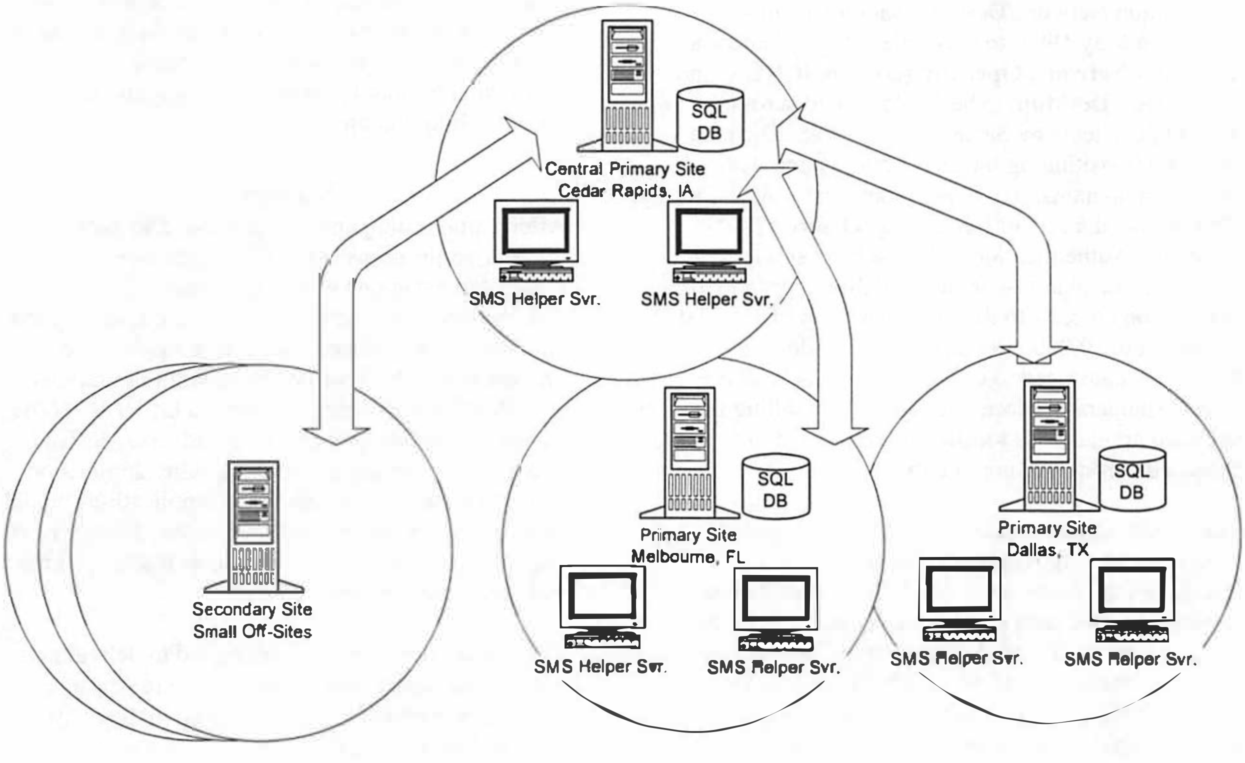
After initial testing and piloting, the distribution of the four applications to the PC clients were to be completed using one of two processes. The distribution processes are labeled Large Application and Small Applications. The Large Application process is mostly driven by bandwidth limitations (i.e. 20 PC clients cannot download Office 97 at the same time on one subnet without adversely affecting network performance). The three other applications were distributed using the Small Applications model. Both processes use the SMS Server model (see next page) to assist in deployment and/or tracking of the software inventory on the PCs.

The distributions were also designed to deliver an unattended installation of each software package, reducing the number of personal installations by technical support staff.

SMS Server Model

Software Distribution Vehicle
Microsoft's Systems Management Server v1.2 (SMS) was chosen as the vehicle for small application deployment. SMS features software distribution networks as well as an inventory function that identifies PCs that need the software package and tracks those that have already received it.

The SMS Server model was set up with the Central Primary Site located at the Cedar Rapids, IA facility. Primary sites are established at locations with more than 250 employees while secondary sites are placed at smaller facilities. Each site was configured to take client hardware and software inventory on a daily basis during the CNOS transition. After the transition was completed, the inventory was then reset to once a week.



SMS Server Hierarchy Diagram

Phase 1: Small Applications

The first wave of software distribution began with the three small applications:

- F-PROT Anti-virus v3.0
- Netscape v3.01
- Adobe Acrobat Reader v3.0.

The SMS Client is the medium in which all small applications are delivered and tracked on the client PCs. The SMS Client was installed during the initial CNOS logon transition process as a normal function of SMS.

Problems & Considerations
Unique to Rockwell Collins are the distributed S: drives (ideal for load balancing) where all software applications reside. To manage software installation points, all installations need to be directed to this common location. A challenge encountered was the SMS installations wanted to map to SMS distribution servers rather than a central drive letter such as S: in Rockwell Collins' case.

Also, on the client side, the Package Command Manager (PCM) maps the next available drive letter to install the delivered software and then removes the drive mapping. The challenge with the vanishing drive mapping comes when uninstalling software or adding additional components. Windows 95/NT requires a user to map to the exact drive letter that was originally used for the initial installation.

Solution
In order to direct all software installations to the central S: drive through SMS, a package was delivered to the user that contained a .PIF file and .BAT file.

Later, the user received a pop-up screen via the PCM. The delivered package was highlighted and then executed by the user. Upon execution, the .PIF file was launched. The .PIF called a .BAT file that launched a Perl script. The Perl script started the installation:

1. PCM -> .PIF (.PIF called when the Package Command Manager's *Execute* button is clicked)
2. .PIF -> .BAT (.PIF file calls the .BAT file)
3. .BAT-> Perl Script (.BAT file calls the Perl Script [i.e. S:\perl\bin\perl.exe S:\scripts\netscape.pl])

The .PIF file was needed to launch the .BAT file because the .BAT file alone would open a window on the desktop that would not close and would hang an installation during any called system restarts.

The .BAT file was called next because a .PIF file would reliably start a Perl script.

The Perl scripts, located on the network S: drive, were used to wrap around the installation processes to provide necessary system modifications such as registry changes specific to Rockwell Collins and to make the installations more unattended than originally intended. No drives are remounted because drive S: is a permanent drive mount received at login and all installs were then received from the S: drive.

Implementation
The distribution of Netscape and Acrobat Reader were done within one Perl script on the first cycle of days as shown below. F-PROT followed the same schedule, starting a week after the initial Netscape push.

- Day 1-3: 100 Users per day
- Day 4-6: 500 Users per Day
- Day 5: 4,000 Users
- Day 10-20: Re-issue rollout to those who did not accept the first time.

Queries were built within SMS to track completions of successful installs and to re-send packages to those who had not installed the software. Typically, a window of 48 hours was given to the user to execute the installation package. If the user missed the window of opportunity then they were re-issued the package on follow-up pushes. These pushes were not mandatory.

The small applications did not present a problem in regards to network bandwidth. Netscape, with a loadset of 24MB, took about 2-5 minutes to install, depending upon the network load and the PC client's hardware configuration.

Phase 2: Large Applications

The second phase of software distribution was delivering Microsoft Office 97 Professional to 6,000+ users. The strategy required a more varied approach compared to the delivery of small applications.

Considerations regarding the installation of Microsoft Office 97 Professional:

- *Space requirements*
  The Rockwell Collins customized installation of Microsoft Office 97 required a minimum of 180MB of available hard disk space.
- *Network bandwidth*
  Testing determined that only four installations per sub net could be running at one time before network efficiency was hindered.
- *User-friendly installation*
  Once launched, the installation needed to be completely unattended.
- *Installation points to be directed to the central S: drive.*
  A common installation point needed to be available for add/remove components, reinstall and uninstall Office 97. All installs needed to be directed to this common point and not to the original installation location.

Based on these considerations a two-tiered approach was implemented. The first step of installations was via CD-ROM, the second step consisted of a metered network installation from an applications folder off the Start menu.

Step 1: CD-ROM
SMS Installer creates a self-expanding executable that repackages one or more software applications into a single package to be distributed and installed automatically. SMS Installer eliminates the need for manual intervention and allows for customizations, including registry modifications. Using SMS Installer, a customized Office 97 installation was created to fit the needs of the Rockwell Collins user community. The executable created by SMS Installer contained over 7,000 registry changes including a few custom registry changes. Some of the customizations and registry modifications in the Rockwell Collins load set included:

- *Common installation point*
  This modification changed the registry to reflect the network path where Office 97 resided. This allowed the user to reinstall add/remove components and uninstall Office 97 without providing the CD-ROM.

- *ClipArt Extra installation point*
  This modification placed the ClipArt Extra installation point in the registry and added a ClipArt Extra Install option to the Add/Remove Programs list, allowing the user to easily install additional clipart.

- *Non-Intervention installation*
  This option was chosen to provide a user-friendly installation for the user community.

The installation script created with SMS Installer was placed on a CD-ROM and tested on a pilot group. After testing was completed, 1,000 CD-ROMs were created and distributed to points of contact throughout the organization. The points of contact were responsible for distributing the CD-ROMs to users within their workgroups for installation.

To prevent license violations, the CD-ROM installations required two criteria, a network connection check and a "flag" file check. The criteria checks were added to the executable through SMS Installer. If these criteria were not met, the installation was terminated.

The use of CD-ROMs decreased the installation time dramatically and no network resources were used. New Pentiums completed the installation in less than five minutes, while older 486 machines averaged approximately eight minutes. If disk space availability was an issue, the installation notified the installer of the additional disk space that was required to continue the installation. Once the installation was launched, no user intervention was required.

More than 1,000 Windows 95 machines had installed Office 97 after the first full week of the CD-ROM release. The CD-ROMs could only be used on Windows 95 machines. The Common NOS configuration changes for Windows NT machines had not been completed when the CD-ROMs were created. As of June 1st, over 6,000 Windows 95 machines have Office 97 loaded locally.

Step 2: Metered Network Installation
A common installation point was needed to add/remove components, reinstall or uninstall Office 97. In addition, it was needed for those in the user community that did not have a CD-ROM drive. Using the Microsoft Select CD, an administrative installation was placed on the network. A custom setup configuration was created for the administrative

installation using Microsoft Network Installation Wizard. Like SMS Installer, Network Installation Wizard allows for customization and registry modifications.

Because of network bandwidth constraints, the network installation was limited to four per sub net at one time by using a metering utility. When the network installation was initiated, the user's host name and sub net were logged to a file on the network. If the metering log file contained four entries with the same sub net as the user attempting to initiate the installation, the user was notified to try again later. After completion, the metering utility cleared the user's information from the metering log file.

Following the metering log, the network installation checked for available disk space and provided appropriate options based on the space available. The three installation options available are:

- If more than 250 MB of disk space was available, the user received a local installation of Office 97

- If only 200 to 250 MB of disk space was available, the user had the option to clear additional space for the local installation or to install a local version with shared applications running from the network.

- If only 100 to 200 MB of disk space was available, the user had the option to clear additional space, install on an alternative local drive, or run Office 97 from the network.

Disk space values were derived from the amount of space required for Office 97, plus additional space required for Windows efficiency and data storage.

Finally, the Office 97 setup was launched using a custom command line. The command line called the setup executable along with the customized configuration setup file. It also contained the necessary switches to perform an unattended installation.

SMS's primary role in the delivery of Office 97 was for tracking inventory of installations. SMS was tested for delivery of Office 97 but was used only on an occasional basis for delivery because of the lack of a network-metering feature. Furthermore, the .pdf file that was provided with Office 97 did not allow for certain modifications that were needed in the Rockwell Collins load set.

Remote Dial-up Machines

A minimal CNOS configuration exists that mounts limited drives and does not run the SMS clients. This configuration is for users who connect to the Rockwell Collins network via modem connections.

Users, however, can still connect to Drive S: via PPP connectivity and SecureID Tokens and download compressed versions of the Small Applications to the client. Large applications are still distributed via CD-ROM.

Summary

Pros

There are other large-scale solutions that are available to perform software distribution, inventory tracking and other tasks. Many also have the ability to perform from a remote connectivity basis. However, these other solutions have a considerably larger price tag than SMS Server does.

Rockwell Collins chose to implement SMS Server as the baseline for Software Distribution using a local CNOS strategy. This strategy also incorporates critical load balancing techniques in the solution.

Customizing specialized installations are made possible by combining the power of Perl with SMS. Along with this technique, an SMS tracking feature can be used for inventory purposes.

Cons

SMS Software distribution is a push technology. Installs can be mandated, requiring the user to stop all work and install the applications immediately or be given a choice for installation within a specified period of time. If the user does not install the application within the specified period, the application must be pushed again by the administrator. For Windows 95 users within a large environment, both choices have downsides.

Also, SMS does not provide metering on a network basis, making Office 97 installations impractical on a large, enterprise network.

For Windows NT, this is not as much of a problem, because the installation can happen in the background.

Rockwell's current setup will inventory a machine once per week. The database is additive, so the burden of cleaning up old or legacy data resides with the Server Administrator.

Acknowledgements

A successful rollout of applications to thousands of PC Clients cannot happen without successful Network, Server and SMS Server design and implementation strategies.

A large part of this recognition goes to the NT Server (Kevin Sizer, John Cole, Bill Ryder and Bill Hittenmiller) and UNIX Server (Bud Lande, Mark Hunt and Troy Page) teams who were responsible for providing the SMS Server infrastructure and testing and implementing the TAS solution.

John Cole, Danielle Legg and Kathy Kraft made up the original Software Distribution team. This team designed, piloted and rolled out the four, locally installed applications to over 6,000 users within a 5 month window.

Netscape is a trademark of Netscape Communications. UNIX is a trademark of X/OPEN. Windows 95/NT, Office 97 and SMS Server are trademarks of Microsoft Corporation.

Author Information

Cam Luerkens is a Senior Engineer within the IT department at Rockwell Collins, Inc. He is the Team Leader for the PC Desktop Integration team. Cam was co-lead, with Kevin Sizer, for the Common NOS and Common Desktop transition project.

John Cole, after serving on the Desktop Integration team, is now an NT Server administrator within the IT department at Rockwell Collins, Inc. His current work is leading much farther into the design and maintenance of the SMS Server Software.

Danielle Legg is a Desktop Administrator within the IT department at Rockwell Collins, Inc. Danielle's main work continues to be in the area of Software Distribution.

References

Wall, Larry, and Schwartz, Randall L.,
with Stephen Potter
Programming Perl
Second Edition
O'Reilly and Associates, Inc., Sebastopol, CA
September 1996: Second Edition

Microsoft Press,
Supporting Microsoft Systems Management Server
Division of Microsoft Corporation
One Microsoft Way, Redmond, Wash
98052-6399

Compaq's New Engineering Compute Farm Environment: Moving Forward with Windows NT

Don Brace *(Don.Brace@compaq.com)*
Andrew Gordon *(Andrew.Gordon@compaq.com)*
Scott Teel *(Scott.Teel@compaq.com)*

Compaq Computer Corporation

Abstract

In the past, Compaq's design and verification distributed compute farm consisted of UNIX-based often times expensive proprietary hardware and operating systems. In addition, internal developers created site-specific load distribution software for the management of design verification compute intensive processing jobs. This environment offered Compaq design engineers (DEs) a finely tuned site-specific environment, where the benefits included on-site developers providing quick user-required modifications, in addition to a mature operating system environment. The downside included higher hardware purchasing and maintenance costs. However, with the emergence of Windows NT, Platform Computing's *Load Sharing Facility* (LSF) for Windows NT, the availability of Electronic Design Automation (EDA) applications for Windows NT, and high-end Intel-based workstations, a Compaq-based Windows NT LSF compute farm proved to be a viable solution. Compaq's internal systems engineers (SEs) and application engineers (AEs) began the process of developing a Windows NT-based compute farm environment that would provide a robust distributed compute farm to be used to facilitate the designing and verification of future Compaq products. This paper describes the issues encountered and the solutions required to bring the Compaq-based Windows NT LSF compute farm online and into production.

1. Compaq's UNIX-based Compute Farm

Compaq's compute farm evolved over a period of time dynamically changing to meet the needs of its DEs. The path to the current compute farm configuration was built with hard work and innovative ideas that have helped to make it an operational success. Compaq DEs have come to rely on its stability and robustness to process their EDA designs.

1.1. History and Motivation

Components used in designing a computer system were simulated in software, starting an evolution in computer design. DEs were able to select components from a database and assemble them into simulated boards. These simulated boards could then be tested with other software, reducing the number of times that the designs needed to be sent to the board manufacturers. This in turn reduced design costs and shortened the design cycle.

Before compute farms, each EDA DE used powerful UNIX workstations to run the EDA applications. DEs soon outgrew their workstations requiring more memory and faster CPU speeds. It was not cost effective to replace each and every desktop workstation, so the next step was to purchase powerful servers. Each DE X-termed into one or more of their departmental servers to run their EDA applications. Frequently, these servers became overloaded, drastically reducing job throughput and slowing down the engineering design cycle. Conversely, there were times when these servers went unutilized by those assigned to them when others could have used them. The departmental server problem grew worse over time as EDA applications demanded more resources.

Attempts were made to do some rudimentary load balancing, meaning DEs called each other and reserved time slots on the servers. This was only a partial solution since problems often arose that caused them to overrun their time slots. The next step was to either purchase or design software, which would manage access to the departmental servers allowing for greater job throughput. Figure 1 illustrates a time-line that depicts the historical evolution of Compaq's EDA environment.

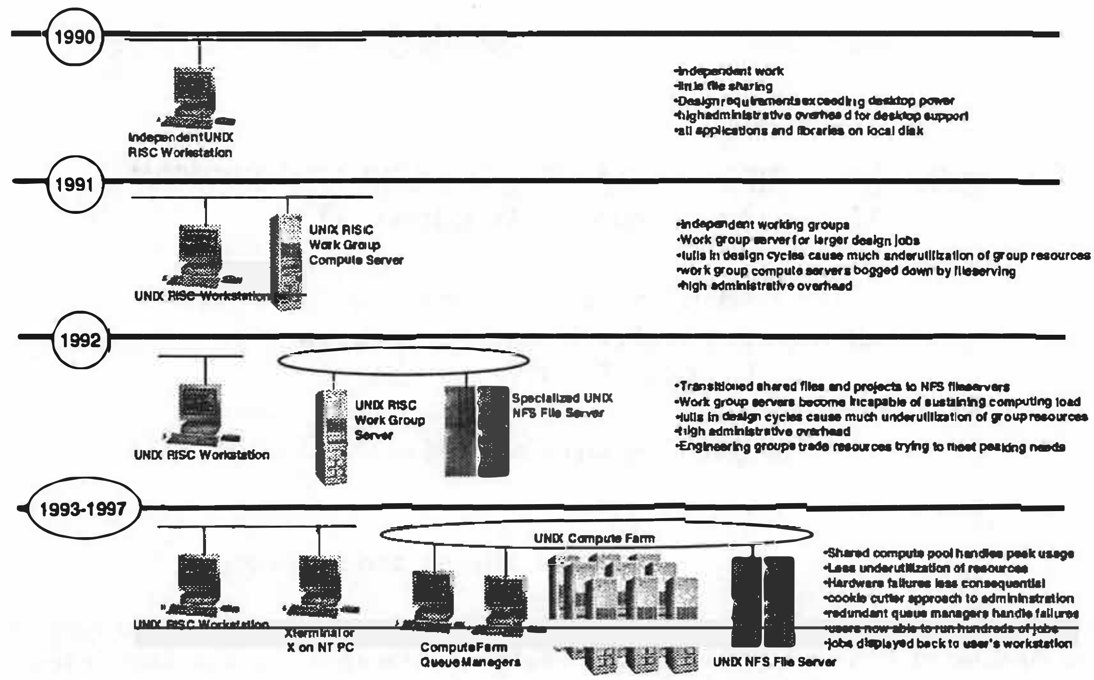


Figure 1. Evolution of Compaq's UNIX-based Compute Farm

1.2. Why UNIX

EDA applications started and have continued on UNIX-based systems at Compaq for the following reasons:

- UNIX was the best-supported OS by EDA vendors.
- UNIX workstations provided high-resolution graphics.
- UNIX supported a large memory model.
- UNIX adopted the X-protocol for GUI support.
- UNIX adopted the Network File System protocol (NFS).

Early on UNIX-based workstations were configured with high-resolution graphics and a larger memory capacity than a normal DOS PC. Later on both the NFS protocol and the X-protocol added functionality in the areas of filesharing and distributed graphics.

Due to enhancements provided by UNIX vendors, often times it was easier to maintain a homogeneous set of hardware platforms and operating systems rather than a heterogeneous one. Once a specific hardware platform and operating system was chosen, it was much harder to adopt another.

Each UNIX vendor selected a different CPU architecture with which to run their UNIX operating system on. This meant that application vendors had to port their applications to each hardware and operating system type.

1.3. In-house Developed Batch Queuing System

At the time job management systems were researched, there were no viable third party solutions that met Compaq's needs. Thus an in-house solution was developed with the following requirements:

- Provide DEs with the ability to submit an unlimited number of both batch and interactive jobs.
- Provide secure access to the batch queuing system.
- Balance the compute farm load.
- Create a virtual environment for each job.
- Create a fault-tolerant batch scheduling system.
- Provide fair access to the compute farm.
- Provide an efficient administrative interface.

DEs typically ran regression tests that amounted to thousands of job submissions to the queuing environment. DEs automated their job submissions since the command line interface was developed to support this functionality. Jobs that ran in the batch queuing system ran with the engineer's UNIX identity, thus protecting the project information.

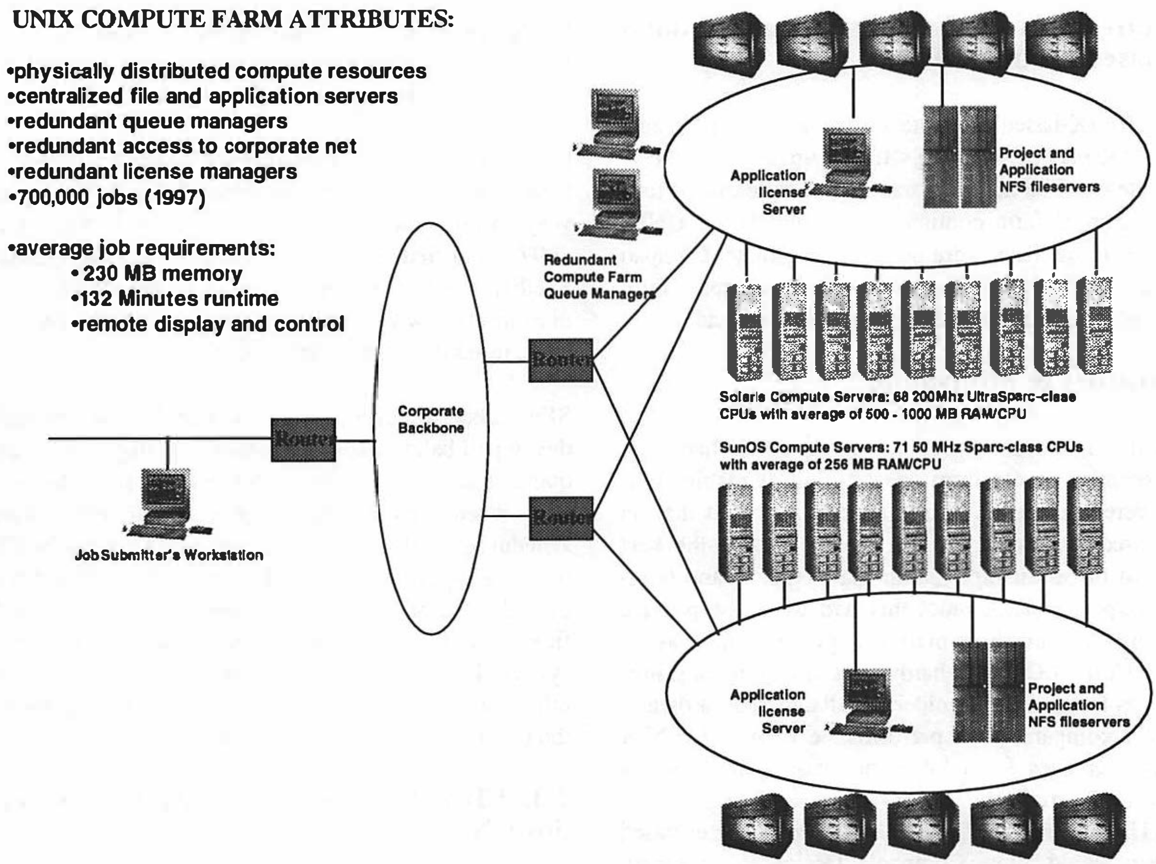


Figure 1 Compaq's UNIX-based Compute Farm

Load balancing was an important feature of the batch queuing system. From benchmark testing of the actual applications, it was found that running one job per CPU maximized job throughout. Also, the scheduler tracked the amount of memory currently in use on each compute server, weighted by the projected amount of memory its running jobs would need over their lifetime.

The batch queuing system used two machines to provide instant fail-over. The scheduling mechanism was a rather complicated token fairness algorithm. DEs were assigned to various projects and the goal was to avoid giving one project priority over another. Ties were broken using round robin. The token costs were determined by benchmarking the compute servers and giving higher token costs to the faster machines.

The administrative interface was GUI based and allows modifications to be made to the running system. Users could be added and deleted from queues and projects, and machines and queues could be added or deleted from the system.

1.4. Compaq's Current UNIX-based Compute Farm

The UNIX compute farm has been in use for over five years and has been quite reliable. In 1997, over 700,000 jobs were run through the UNIX compute farm. The compute farm consists of a collection of Sun, Tatung and Solbourne compute servers and two major user queues. One queue is for batch oriented jobs and the other queue is for interactive jobs.

The compute server hardware consists of 3 Ultra Enterprise 4000 machines each with 8 GB of memory, 21 Tatung Ultra-2 machines each with 1.2 GB of memory, 15 Solbourne 900 machines with 1.2 GB of memory, and 2 SPARC 20 machines with 128 MB of memory. The Ultra class machines use Solaris 2.5 as the operating system and the Solbourne and SPARC 20s run SunOS 4.1.3_U1. In total, there are 142 job slots available for DEs to use. The entire batch queuing management system is managed from 2 Sun SPARC 20 machines.

2. Motivations for Establishing a Windows NT-based Compute Farm

As the UNIX-based compute farm matured, it became an integral part of Compaq's time-to-market engineering strategy. The memory size and complexity of jobs in the compute farm continued to grow. Older UNIX resources in the farm were becoming obsolete. Compaq SEs needed to continue the success of the compute farm while reducing costs and administrative overhead.

2.1. History & Motivation

When the time came to replace aging UNIX hardware and expand compute farm capacity, all available solutions were considered. While reports indicated that an Intel-based hardware platform would provide the best total cost of ownership, easiest maintenance, and highest price/performance ratio, this had yet to be proven. Selecting a hardware platform proved the easiest choice. Current Compaq hardware ran engineering jobs 2-3 times faster than the older SPARC-based hardware, and was comparable in performance to newer UNIX hardware choices for a lower purchase cost. Memory prices were similarly competitive. Rejecting RISC-based UNIX hardware for the less expensive Intel-based platform would allow Compaq's Design Environment to expand compute farm resources.

When considering operating systems (OSs), Solaris led the pack. Sun's Solaris X86 was available for Compaq hardware, was a mature product, and had the advantage of easing the task of porting administrative and engineering scripts and utilities. However, Microsoft's Windows NT boasted solid support from a growing number of engineering application software providers, which had no plans to port their tools to Solaris X86. Encouraging results from initial internal UNIX to Windows NT porting efforts established a confidence level that applications, scripts, and utilities could be ported. Hardware, operating system, and EDA applications indicated Windows NT as the OS of choice. It became clear that Intel-based hardware running Microsoft Windows NT would become the core of Compaq's future engineering computing farm.

2.2. Off-the-Shelf Load Sharing Product

Though Compaq's DEs had been enjoying the benefits of a locally developed load balancing, fault-tolerant job queuing system, the cost of those benefits were being weighed against the purchase of an off-the-shelf product. The development, testing, and support resources required to produce a system specifically tailored to Compaq's business, computing and engineering needs were growing, directly in contrast to the compute farm goals of shrinking administrative overhead.

Compaq required a commercially available, supported product capable of being configured to meet Compaq's very specific needs. A research project starting in early 1997 identified Platform Computing's Load Sharing Facility (LSF) as the only product on the Windows NT platform that would meet Compaq's needs for performance, reliability, flexibility, and support.

SEs looked at several products including an internally developed batch scheduler before selecting LSF's batch queuing software for Windows NT. Most of the products where mainly built for calendar event-driven scheduling, and our requirements could not be fulfilled by these types of products. However, the LSF software provided all of the basic requirements and most of the flexibility of the Compaq-developed compute farm system. LSF would leave SEs free to develop and tune other parts of the environment without having to port the internal batch scheduling system.

2.3. EDA Applications Migrating to Windows NT

As the Windows NT compute farm plan developed through the first half of 1997, EDA software vendors announced their support for the Windows NT platform. By August of 1997, a key application in ASIC chip design became available on Windows NT, increasing the desire for a Windows NT-based computing farm. Cadence Design Systems' Verilog-XL, the main EDA application in use in the UNIX-based compute farm, was now supported on Windows NT, and exhibited comparable performance to its UNIX-based cousin. Eventually, other EDA software tool vendors would announce support for the Windows NT platform.

Adding a Windows NT Compute Farm would expand Compaq's engineering computing capabilities without the purchase of additional non-Compaq hardware. It would make more job processing power available in the UNIX compute farm by allowing some work to shift into the Windows NT farm. Cost savings would be realized in initial purchase, hardware maintenance, support, and local development. Engineering application support was growing.

3. Setting up a Windows NT Compute Farm: Investigation, Testing, and Planning

After an initial internal proof-of-concept evaluation for a Windows NT compute farm using LSF, Compaq SEs recommended to continue on with an in-depth evaluation. SEs continued to evaluate the Windows NT compute farm concept extensively in addition to laying the foundation for a production-ready compute farm through investigation, testing, and planning. Due to the lack of experience with Windows NT in a batch-processing application environment, Compaq SEs worked through each of the following areas recording their results during each phase.

3.1. Defining the Compute Farm Environment Requirements

At the highest level, the project plan defined the major compute farm requirements:

- Compaq Hardware
- Windows NT-based Operating System
- Configurable Fault-tolerant Batch Scheduling Mechanism
- Real Time and Historical Usage Reporting System
- Distributed Graphics for Interactive Jobs
- Remote Capabilities for Dial-Up Users
- Remote Administration Capabilities

One of the most important goals of this project was to use Compaq-based hardware during the EDA design verification phase of Compaq's products. In a search for the right hardware, SEs evaluated all Compaq-based hardware for price/performance, and its processing granularity to reduce the compute farm's single point-of-failures. The Compaq Workstation and ProLiant products fulfilled these requirements.

Due to an existing user installed base of Windows NT on Compaq-based products and the availability (or planned availability) of EDA applications on the Windows NT platform, SEs agreed that Windows NT would be the operating system of choice.

Like the internally developed scheduler, the Windows NT compute farm scheduler would need to provide flexible configuration parameters to allow for the implementation of site-specific scheduling policies. Due to the criticality of internal schedules of EDA verification projects, the batch scheduling mechanism would need to provide the following functionality:

- Easy configuration capabilities.
- Fault tolerant across network or server failures.
- Scheduling based on resource needs such as memory.
- Load-balancing.
- Access control to batch queuing system.
- Access control to administration commands.

Once the compute farm reached production status, compute farm usage statistics, both real-time and historical, would provide critical resource planning information. EDA management defined the need for usage reports, both real-time and historical, for this compute environment.

During the initial stages of the ASIC verification process, engineers debug large verification jobs working out problems in both the design and the regression tests. The debugging process requires an interactive session with the regression software. As such, if an engineer wanted to debug a verification job using a compute farm resource, the engineer would need to interact with the remote job. Historically, some sort of distributed graphics mechanism provided this functionality. As a result, EDA engineers identified the need for interactive capabilities during large regression setup and debugging from within the compute farm.

Once ASIC verification engineers have worked through problems within their regression suite, they will submit a large set of regression jobs to the compute farm. This can be literally hundreds or even thousands of jobs per submission setting. Often times during time-critical regression, engineers will need to log in from off-site locations and check the status of a regression suite. As a result, engineers identified the need for remote access capabilities including job submission and job monitoring.

As with any distributed computing environment, there exists the need to administer machines in an efficient manner, especially when machines are physically located in separate locations. Simply, the time of traveling to the different computing facilities can be expensive.

In addition, after the compute farm reached production status, hotline SEs who are normally tied to a phone line would need to remotely log onto the problem compute server and understand the cause of any problems. This was a definite requirement.

3.2. Selecting Initial Windows NT Compute Farm Applications

EDA AEs identified an initial set of design verification tools to be used within the Windows NT compute farm based on several factors:

- Availability on Windows NT
- Ability to run in batch mode
- Functionality during the verification process

Not only did the EDA applications need to be available on Windows NT, but certain internal verification libraries would need to be ported over to Windows NT. These modules are linked in at runtime. Furthermore, the verification tool would have to have the option to run in batch-mode. After identifying a small set of possible EDA applications, Verilog-XL would become the first application to run in the compute farm environment.

In order to run any EDA application on a Windows NT machine, the environment had to be setup correctly, and once EDA AEs establish a working environment, this environment would need to be transferred to the compute farm server at job submission time. During the initial application setup, AEs chose to install the EDA applications onto a centralized file-serving setup in which application binaries and libraries would reside in a single file-serving location. This would reduce the administration burden.

To help facilitate the job submission process, EDA AEs developed submission tools that submit jobs during the design verification process. A portion of these tools required the batch job to run from the submission (network drive) directory—this was added to the list of application requirements.

3.3. Researching the Major Issues

Having defined the requirements of the Windows NT compute farm environment, project members turned to discussing the issues that needed to be resolved. The following list enumerates the major issues that had been identified before and during the initiation of the compute farm project:

- Identifying the optimal compute farm hardware configuration (compute model) for our EDA applications: #CPU's/machine, Memory/CPU, network bandwidth.
- Identifying an initial optimal LSF batch-queuing configuration setup: #queues, scheduling parameters.
- Identifying an optimal UNIX-NT file sharing solution for our environment to access existing project information on UNIX-based fileservers.
- Identifying a flexible remote administration tool to allow remote access to compute servers used during administration or debugging.
- Identifying or developing a dynamic Windows NT drive-mounting mechanism needed by in-house developed EDA application tools.
- Identifying and reviewing the Windows NT file serving infrastructure configuration and setup.
- Identifying EDA application environment problems.
- Testing GLOBEtrotter's FLEXlm license mechanism from Windows NT using UNIX-based license servers.
- Identifying and installing UNIX-based development and scripting tools within the centralized Windows NT file-serving environment.
- Identifying administration tasks that could be automated.
- Identifying administration and support documents that needed to be developed.

3.4. Extensive Evaluation and Testing

To identify or resolve certain issues in addition to verifying previously made decisions, SEs spent a considerable amount of time testing, especially with regards to the LSF products, UNIX-NT file sharing products, and EDA applications. The following sections describe the test results.

3.4.1. LSF Product Evaluation and Testing

After an in initial proof-of-concept evaluation, the SEs identified the LSF batch-scheduling product as the only real possibility other than porting their internally written job scheduler. However, even though Platform provided a flexible batch queuing mechanism on Windows NT, SEs needed to extensively test the product, trying to understand its strengths and weaknesses especially on the Windows NT platform. As such the SEs prepared an extensive test plan to exercise the LSF product in the following areas:

- Administration, Manageability, & Usability
- Fault-tolerance
- Reliability
- Batch Scheduling Features and Capabilities
- Platform Computing's Support Capabilities

3.4.1.1. Administration, Manageability, and Usability Evaluation

Administration and manageability of a LSF cluster was fairly simple after completing the initial setup and configuration. Initially, the LSF 3.0 installation was not wrapped up inside of a simple Install Shield mechanism as most Windows-based applications, but Platform added this type of capabilities with the LSF 3.1 product. The administrative burden can be reduced when application binaries and configuration files are centralized. The LSF product incorporates this feature quite naturally having ported their product from a UNIX environment.

For administrators, the LSF product provides an administrative command suite that can stop, start, restart, and shutdown each respective service and/or daemons. The most difficult part of the Windows NT LSF administration is administrating the Windows NT itself. The limited number of debugging tools makes problem solving more difficult.

One administration example is adding a machine to the LSF cluster. Although there is no current Windows NT GUI that provides an editing interface, the process of adding a machine can be summarized in about five steps, each step being intuitive for a knowledgeable systems administrator.

From a user perspective, most LSF commands mimic the functionality of those used within the internally developed batch queuing system. In addition, the LSF product contains some functionality, such as the ability to view the history of a job. A job history provides historical job information: how long a job waited, the LSF server it executed on, the final exit status, along with other useful information.

3.4.1.2. Fault-tolerance Testing

SEs tested LSF's ability to recover from simulated network outages, simulated cluster failures, and simulated network file system failures. SEs simulated a network outage by unplugging the network connections to several LSF cluster machines. Likewise, members simulated a cluster failure by powering down the currently running master LSF cluster server. SEs simulated file system failures by shutting down the appropriate file share server.

Since there are two LSF layers—the LSF Base and Batch—one requiring the services of the other (Batch requires the services of the Base software), recovering from major network or cluster errors occurs in two phases. First, the *lim* (LSF Base software) daemons must determine if the master *lim* has gone away and if so, pick a new one. Secondly, the *sbatchd* (LSF Batch software) must determine if a master batch daemon has gone down, and if so, pick a new one. The *lim* daemons are considered functional once the *lshosts* commands reports a full list of LSF cluster servers, while the *sbatchd* daemons are considered functional after the bhosts command reports a full list of LSF batch cluster servers.

While either the *lim* or *sbatchd* layers are non-functional, both user commands and normal scheduling activities will cease; however, previously scheduled user jobs will continue to run. User commands will continue to try and contact the master scheduler until a specified time-out valued has been reached. If the time-out value is reached, the user command exits with a failure; otherwise, the user command will complete the task when it is able to finally contact either the master *lim* server or the *mbatchd* (depending on the type of user command). Tests indicate 30 to 45 seconds cluster reorganization. During reorganization, no user *lim* or *sbatchd* commands will complete successfully until each respective layer has regrouped itself.

3.4.1.3. Reliability Testing

With most out-of-the-box products, there exists certain stress conditions under which the application will stop functioning in a production quality manner. Knowing these conditions can be very beneficial to system administrators or managers. SEs essentially tried to find out how to break LSF. One such example was the MBD_SLEEP_INTERVAL parameter located in the lsb.params configuration file. Setting this interval too small causes the master batch server to spend too much time trying to schedule jobs, unable to respond to new requests from client machines.

3.4.1.4. Batch Scheduling Features Evaluation

SEs experimented with several scheduling parameters available in the LSF Batch software. In general, all parameters worked as expected. Jobs were scheduled on the correct resource based on the requested resource requirements. For Compaq's design verification computing environment, a batch scheduler's memory management capabilities are critical to efficient memory and CPU utilization. LSF provides this by means of a resource requirement selector with different selection possibilities. SEs implemented the memory resource requirement selector with a *linear decay duration value* specified at job submission time by either the user or a pre-determined default value. The *linear decay duration value* indicates how long the job will execute before it has consumed its total memory requirement. Initially, the default value will be set to 30 minutes, but SEs urged users to set this value using a command line option

SEs tested the *Dispatch Window* mechanism by running user jobs on certain team member's desktop machines at night. SEs configured these desktop machines to accept jobs from 7PM until 5AM the next morning. This feature worked, but SEs do not plan on implementing this feature in any initial release due to certain manageability issues such as remote administration and controlled downtime.

Fairshare testing pointed out the possibility of one group being temporarily starved of slots due to their past excessive use of the cluster. The window of accumulated CPU and wall clock time is a configurable option in the lsb.params file. Although the *Fairshare* mechanism does not exactly mimic the *Token Fairness* algorithm used within the Compaq's internal-developed scheduler, it does provide a fairness policy based on user groups. The LSF scheduler implements *fairness* by setting up "fair-shares" which indicate how much usage units should be given to each user or user-group. The scheduler keeps track of past CPU and wall-clock job usage, and uses this information on future scheduling decisions.

Finally, SEs tested the scheduler's ability to scale by increasing the number of nodes in the cluster and repeating some of the earlier tests. Even with 10000 jobs queued up, the scheduler continued to respond to client requests in a constant predictable time.

3.4.1.5. Platform Computing's Support Capabilities

SEs tested Platform Computing's ability to provide quality support in a timely manner. Project staff concentrated on four specific types of problems often encountered within a production-computing environment: *high-priority installation problem, dead-in-the-water problem, enhancement request*, and *a possible bug problem.*

For all test cases, Platform Computing provided quick support response, usually resolving within 15 minutes. Due to the small size of the company, SEs could quickly access the actual LSF code to determine if a bug existed or if we had accidentally configured LSF incorrectly. And if online SEs could not resolve the problem, LSF support members would summon the Windows NT developers to the phone for help.

3.4.2. UNIX-NT File Share Testing

Since Compaq's design files resided on UNIX fileservers, it was important to identify a method for sharing those project files with Windows NT systems. Making copies to an Windows NT server and keeping them synchronized was not an option, nor was using ftp to transfer files as needed. A true filesharing solution enabling simultaneous access from both platforms was required.

Filesharing options are limited to either client-side NFS or server-side SMB/CIFS interpreters. The PC NFS client products allow an Windows NT PC to connect directly to remote NFS filesystems using NFS protocol through an add-on protocol stack. The server-side solutions are SMB interpreters, which cause the NFS fileserver to appear as an Windows NT server to Windows NT clients. None of the filesharing products have coordinated file locking between Windows NT and UNIX, so there is some danger that files may be corrupted or overwritten if accessed at the same time by both platforms. Since file-naming conventions are different between Windows NT and UNIX, there are also some

things to look out for in this area. Some files created with perfectly legal filenames in UNIX become inaccessible from Windows NT. Case sensitivity is an issue that has been worked around in most products, but not solved. Filesharing users must be aware of the limitations of each platform's filenaming conventions. Windows NT and UNIX have different security ID and file permission models, so there were issues here too.

The filesharing product evaluation revealed that there was no product yet that provides a seamless file sharing solution. It would be up to engineers and administrators to use filesharing carefully, and avoid the known problems until better solutions become available.

3.4.3. EDA Application Testing

EDA AEs spent considerable amount of time testing Windows NT EDA applications both on the their Windows NT desktops and also in the Windows NT compute farm. It was not until just recently that certain EDA vendors had started porting their tools over to Windows NT and certain bugs were still being worked in parallel to bugs in our compute farm environment. EDA AEs first worked to provide a production desktop environment that would allow certain EDA applications to run from a single application server as done historically with the UNIX desktop and compute farm environment.

After EDA engineers worked through desktop support issues, they moved on to getting applications to correctly run in the compute farm. The number one issue centered on making sure the desktop environment was correctly mapped to an LSF server during job initialization. The LSF software appeared to have been written to transfer only certain environment variables based on their interpretation of the Windows NT environment. After working through the environment issues, SEs updated the already internally developed LSF wrapper script to work-around these items.

EDA AEs had developed a suite of submission and verification tools to augment the testing functionality that already existed in the vendor product such as Verilog-XL. As such, internal EDA AEs worked on porting their submission tools over to Windows NT and changing their submission modules to use the new LSF-based job submission mechanisms. Once AEs had worked through issues both from a desktop issue and a compute farm perspective, large scale job submissions commenced and application load testing began to identify any stress-related problems during intense processing by real EDA applications.

3.5. Windows NT Compute Farm Environment Version 1.0

After internal discussion about the compute farm requirements and impending issues, SEs constructed a project plan that defined reasonable project milestones. The initial release of the compute farm as a production system fulfilled only the following previously listed compute farm requirements:

- Compaq Hardware
- Windows NT-based Operating System
- Configurable Fault-tolerant Batch Scheduling Mechanism

SEs will complete the other compute farm requirements in subsequent phases.

4. Compaq's Initial Windows NT Compute Farm Environments

SEs installed and configured all hardware into a climate controlled computing facility. This section describes the actual compute farm setup and current solutions to the previously mentioned issues.

4.1. Compute Farm Hardware

As of the writing of this paper, SEs successfully configured a 60 CPU LSF cluster running Windows NT 4.0 with SP3 cluster consisting of the following list of Compaq Workstation and ProLiant machines (see figure 3):

- 20 Professional Workstation 8000 with 3 x 200 MHz PentiumPro CPUs and 3 GB of total memory used as LSF Batch servers all using 100 Mbs FDDI connections.
- 3 ProLiant 5000 fileservers with 2 x 166 MHz CPUs each with 128 MB of total memory using 100 Mbs FDDI connections.
- 3 ProLiant 2500 fileservers with 100 Mbs FDDI connections.

At the start of this project, the Professional Workstation 8000s provided the largest expandable memory option of up to 3 GB of total configurable memory. As a result, the main core of the compute farm consists of

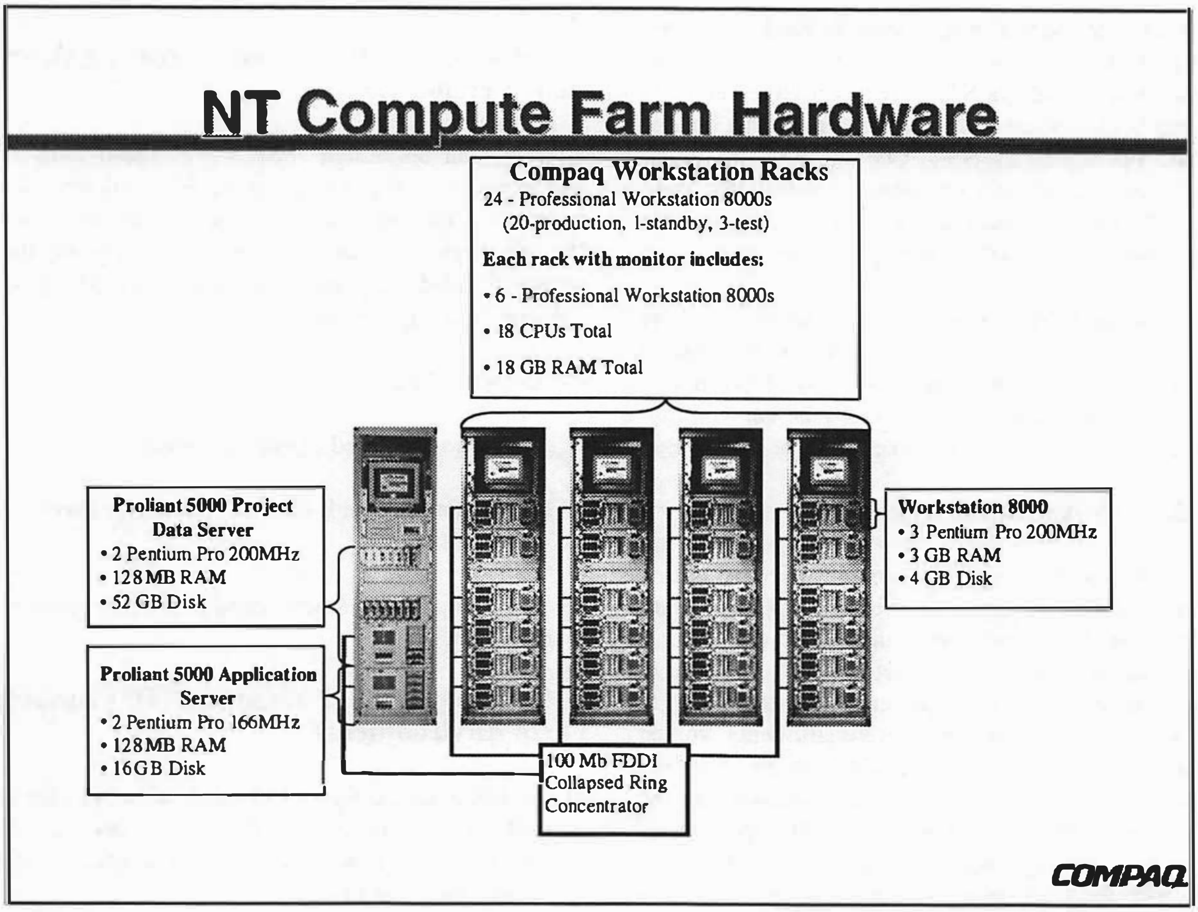


Figure 3. Compaq's Windows NT Compute Farm Configuration

20 Workstation 8000s each configured with 3 GB of total memory using 100 Mbs FDDI connections.

Before the compute farm project, SEs designed a centralized fileserver model for the Windows NT desktop infrastructure. This allowed for EDA applications to be stored and executed from one central location—a similar UNIX environment already existed. The compute farm jobs access binaries, configuration files, and project data located on 2 ProLiant 5000 fileservers—a (primarily read-only) application server and a (read-write) project server.

During the evaluation phase, occasional network, application, and operating system errors caused problems for the master LSF batch scheduler. SEs recommended setting up two separate LSF server machines to be used only as schedulers and not as compute servers, preventing rogue jobs from causing a system failure on the master scheduler.

4.2. Batch Queuing Software

SEs deployed LSF 3.0b version onto the compute farm resources. From an architectural view, the LSF software is cleanly divided into two layers:

- A Load Information Management layer (LIM) discretely manages the load indices of all machines in the cluster.
- A batch application layer uses a LIM API to determine the correct resources to dispatch jobs to.

Each layer provides a suite of commands for viewing or changing the cluster.

4.3. Remote Windows NT Administration

SEs had looked at several different products, including Microsoft's System Management Server (SMS), which was also being tested as an application installation tool. As such, the team tried using the SMS remote control

tool, but the SMS remote control tool would often stop working very early into the remote connection, in addition to being slow in general. Next, SEs started testing PCI Ltd's PC-Duo product. The PC-Duo product had good response along with an access control mechanism to provide some security control. As a result, SEs decided to use PC-Duo as their remote administrative tool of choice.

4.4. Hardware Monitoring

To provide a hardware monitoring mechanism, SEs installed and configured the Compaq Information Management (CIM) agents on all of the compute server resources. The CIM agents provide hardware state information and errors to a CIM management application.

With regards to monitoring in general, SEs setup a separate monitoring machine to run the CIM management GUI application continuously. In addition, several other monitoring programs run on this designated machine.

4.5. Time Synchronicity

As with most distributed compute farm or cluster systems, synchronized clocks are critical to distributed algorithms. In addition, the application themselves can be time-dependent, relying on a common time-stamp for project files. Furthermore, licensing mechanisms such as FLEXlm rely on a consistent clock between server and requesting client.

In the current UNIX compute farm environment, SEs had already setup xntpd time servers used by XNTP clients updating their system clocks. Since the infrastructure already existed, SEs ported the xntp software code to Windows NT and deployed on all Windows NT machines accessing the compute farm environment. This proved to be a critical element as several problems were traced down to the xntpd client not running correctly on the problem-related compute farm server.

4.6. File System Infrastructure and Application Setup

SEs reduced administrative efforts by centralizing the LSF configuration files. This provided a large administrative gain by having only one location for updates. To simplify initial compute farm requirements, EDA AEs conceded to running jobs that would only access Windows NT files and not UNIX-based files via a UNIX-NT file sharing solution. However, due to the already large investment of UNIX-based file servers, some form of UNIX-NT solution will become a future requirement.

SEs had already established a common primarily read-only application server for EDA applications and supporting tools as done in the similar UNIX environment. In addition, SEs configured several data servers for project-based file storage.

4.7. Administration Scripts and Services

As part of the compute farm production release, SEs developed scripts and services to automate some administrative tasks:

- LSF job output file renamer—this utility changes the output file names to coincide with those generated by an in-house batch-scheduling UNIX-based system. This reduced certain EDA submission tool porting efforts.
- LSF output directory cleanup—this utility would manage the user's job output directory compressing and archiving job files at specific intervals.

4.8. LSF Wrapper Submission Script

Early in the evaluation, the SEs deveoped a *bsub* wrapper script for the following reasons:

- To reduce the LSF administrative burden of LSF by centralizing the file job scripts and output files.
- To implement control measures with regards to resource requirements and usage.
- To implement a dynamic Windows NT file-sharing solution on the LSF servers when needed.

The wrapper script takes the user's job and does the following:

- Checks the command line options for valid memory estimates.
- Checks for valid project names and project access rights.
- Inserts network drive mount and un-mount commands.
- Submits the newly created job via standard *bsub* command.

4.9. Automatic Mounting Utilities

The EDA AEs had created a suite of EDA application tools. When executed in the UNIX compute farm environment, these tools relied on the batch job running from the initial submission directory. As a result, Windows NT compute farm jobs would need to do the same. SEs placed hooks into the LSF wrapper script to automatically mount the network submission drive at job startup, move to the submission directory, and reverse the procedure during job cleanup. One caveat became the ability to cleanly kill a job without leaving drive mounts around—eventually running out of drive letters. Later on SEs looked to a special job killing mechanism to keep leftover drive letters from collecting on a compute resource.

4.10. Windows NT Configuration

SEs examined the Windows NT Workstation vs. Server decision and decided to build compute farm machines using the Windows NT Server product. SEs loaded Windows NT 4.0 server with Service Pack 3 on all machines to be used within the compute farm and left the default performance settings.

During system and application testing using UNC paths, SEs identified a Windows NT Redirector problem. After working with Microsoft SEs, Microsoft provided a patch that fixed the problem and is now applied during the build process for all LSF compute servers and other Windows NT machines (Q179983 and Q179873).

As per the LSF documentation, SEs configured LSF compute servers with static IP addresses. Although our Windows NT environment primary consists of DHCP clients, the persistency of the leases allowed our LSF clients to work successfully within the LSF cluster even though the LSF documentation recommended only static IP addresses.

4.11. License Management

Licensing issues were addressed and resolved before the application could correctly run in the Windows NT compute farm environment, including the batch queuing software itself. EDA applications and LSF's batch queuing software used the industry standard FLEXlm software for license management. This facilitated the initial setup for two reasons:

1. UNIX-based FLEXlm license servers could manage (issue) client license checkouts from Windows NT clients.
2. SEs had already established redundant UNIX-based FLEXlm servers for applications that ran in the UNIX-based desktop and compute farm environment.

As such, SEs and AEs setup the new Windows NT application license features and keys on the already established UNIX-based license servers, thus reducing the overall setup effort.

4.12. Web-based User and Support Documents

Like all products or projects, the final deliverables must include good documentation. For the Windows NT compute farm, AEs and SEs updated support procedures that helped both users and administrative staff to understand the new system. Due to its simplicity and usability, SEs chose to create web-based documentation and place on local (departmental) web-sites for easy accessibility and searching capabilities.

The final documentation included several deliverables including evaluation reports, administrative setup procedures, administrative trouble shooting guides, administrative test procedures, bug tracking lists, user guide, user FAQs, and future project plans.

4.13. Compute Farm Test Environments

Test environments are essential for improving the compute farm without risk to production job activity. As a completely separate environment, the test farm insulates the production farm from harmful events. New versions of utilities, scripts, computer hardware, and even the job queuing and load sharing facility itself may be tested without affecting production jobs. Coupled with a detailed test plan and bug tracking, the test farm environment ensures a stable production compute farm. Operating systems, utilities, applications, and hardware must be kept at the same revision levels as the production environment

Compaq SEs setup several test compute farm environments to allow SEs and AEs to test applications and compute farm tools independently without affecting each other or the production compute farm environment.

5. Future Windows NT Compute Farm Goals

As mentioned previously, SEs had only reached certain compute farm goals, leaving some for future compute

farm releases. Work continues on the remaining initial requirements and some additional goals that will improve the robustness and functionality:

- Intelligent Job Terminator
- LSF Hardware and Software Monitoring System
- UNIX-NT File Sharing Solution
- Job Profiling Service
- Real Time and Historical Usage Reporting System
- Remote Capabilities for Dial-Up Users
- Remote User Control for Interactive Jobs

5.1. Intelligent LSF Job Terminator

The LSF wrapper script created by SEs is treated by the LSF system as a job component, it does not know that it should not kill this process whenever a user kills a running job. Thus the kill command kills the wrapper script as well, leaving behind mounted systems and used drive letters. A local kill command, called *cbkill* was developed as a workaround to this problem. The *cbkill* utility behaves similarly to LSF's kill utility, however it does not kill the wrapper script.

LSF has a rich set of APIs that enabled the *cbkill* utility to obtain the necessary information about a running job from the LSF server. The kill command is sent to the compute server though the use of a DCOM executable that starts up upon demand searches for the target processes and kills them.

5.2. Compute Farm Monitoring System

SEs are designing a compute farm monitoring solution to automate problem identification and resolution within the compute farm. Hopefully, this system will ease the administration burden, allowing administrators to accomplish productive tasks as opposed to fire-fighting compute farm problems.

Initially, the monitoring system will try to identify various system and LSF job-related problems:

- Jobs that are "stuck" or consuming considerably more resources than requested.
- Jobs with password problems. On Windows NT, the impersonation issue is resolved with the Windows NT application storing the user password encrypted in some hidden file and performing the *LogonAsUser* Win32 API call. The service control manager is a good example of a program storing an encrypted password to achieve a separate identification at startup time.
- Available drive letter problems. Currently, the setup wrapper script will mount drives for the job. These jobs can die abnormally and cause leftover mounts to accumulate. The Microsoft Windows Terminal product will help out here.
- LSF daemon problems such as an abnormal termination. Although LSF is fault-tolerant and will bypass a malfunctioning LSF server, SEs want to bring the compute server back into the cluster ASAP.

5.3. UNIX-NT File Sharing Solution

While Microsoft has announced plans for a PC NFS product in an add-on pack for enhancing UNIX/NT interoperability, current plans for the future compute farm call for using a server-side solution. Server-side solutions are easier to administer, debug, and license. Compaq plans to use Auspex Systems' NeTservices product on its Auspex NFS file servers to give engineers access to a common project directory from both Windows NT and UNIX computing platforms.

This strategy will assist engineering teams that are in transition from UNIX to Windows NT, preventing the need for a drastic, all-or-nothing transition from UNIX to Windows NT. Engineering teams that have already completed the transition to Windows NT will have less need of file sharing between platforms, and will have their project and home directories located on Compaq ProLiant fileservers running Windows NT Server.

5.4 Job Profiling Service

Engineers need performance information about their jobs to accurately predict the execution time and resources required for other jobs. A service program that runs on each compute server was created that periodically runs through the list of running processes and records statistical information which is appended to each job logging file. Applications Engineers have written utilities that take this information and store it in a database that is used in setting up job parameters for subsequent job submissions.

The information gathered is basically the same as what the task manager displays, however, only those processes that belong to the job stream are recorded in the logging file for the job. The code for this service was taken from the *tlist* utility found in the NTRESKIT.

5.5. Real Time and Historical Usage Reporting System

The compute farm is currently running real EDA application jobs, and there exists a need for both real-time trending and historical statistics that describe the general usage of the cluster. Historical and real-time statistical reports will be imperative for future capacity planning or current reallocation. LSF provides an LSF Analyzer product that may provide the necessary reports that SEs are requiring. Real-time statistics go beyond just providing a simple snapshot. These need to provide a trending ability to understand the overall load of the cluster over short dynamically configurable interval.

At this time, SEs are beginning to evaluate the LSF Analyzer and its ability to generate specified reports. SEs will begin to define the real-time statistical requirements and determine the need for an in-house developed solution.

5.6. Remote Capabilities for Dial-Up Users

Future plans include providing a dial-up solution for engineers wanting to access the compute farm from home or other remote sites. So far, the LSF software has been the biggest obstacle in that all LSF client and server machines within the cluster must be name-resolvable during configuration initiation for security reasons, which causes problems for dial-in users. Dial-in users do not generate an IP-name map until dial-up time via DHCP.

We are currently looking into several possible solutions:

- Using PC-Duo to allow engineers to access their desktop machines and the LSF compute cluster.
- Getting Platform Computing to make software modifications to allow this type of user to use the compute farm. At this time, LSF cannot handle this type of client machine.

5.7. Remote User Control for Interactive Jobs

DEs have requested the need for a remote interactive job control solution for jobs submitted to the Windows NT compute farm. Providing a solution has been difficult due to Windows NT's desktop only philosophy. Porting UNIX-type tty functionality over to Windows NT has been difficult due to a lack of process signals within Windows NT process model. Currently, SEs are looking at other remote terminal applications, such as Microsoft's Windows Terminal Server, to provide this functionality for interactive compute farm jobs. Microsoft's Windows Terminal Server may support a command line interface, which would allow its terminal sessions to be automatically started from a job stream and incorporated into the LSF software.

6. Summary

This paper describes the solutions required to bring the Compaq-based Windows NT LSF compute farm into production. Reasons for creating a Windows NT LSF compute farm include cost-effectiveness, migration of EDA tools to NT, and processor performance levels comparable to UNIX RISC machines.

Aging UNIX hardware and the need for expanded compute farm capabilities at a reduced cost triggered a series of hardware evaluations, seeking to upgrade existing platforms or adopt another. These evaluations resulted in the adoption of Compaq hardware running Windows NT. Lessons learned from UNIX-based compute farm were brought over to the Windows NT compute farm. Presently, the Windows NT compute farm consists of Compaq hardware, Platform Computing's LSF software, remote administration tools, and the main EDA tools that DEs use in the design process. Some issues remain and will be resolved in the future through projects that are in place to provide required solutions.

Compaq, ProLiant, Professional Workstation 8000, registered U.S. Patent and Trademark Office. Product names mentioned herein may be trademarks and/or registered trademarks of their respective companies. ©1998 Compaq Computer Corporation. All rights reserved. Printed in the U.S.A.

Designing an Optimized Enterprise Windows NT/95/98 Client Backup Solution with Intelligent Data Management

Kevin M Workman, Earl Waud, Steven Downs, Mikel Featherston
Qualcomm Inc.

Abstract

Recent times have seen a vast increase in the number of PC's deployed on company networks, as well as a significant increase in the drive space attached to those machines. This situation presents a growing problem for those responsible for both maintaining corporate networks and insuring the integrity and safety of corporate data. The increase in the enterprise data set translates to an increase in the amount of data on the wire, as it is sent over the network to be archived, as well as an increase in the amount of tape media that must be used to protect the data. Faced with an ever-increasing amount of data to support, it is becoming necessary to discover a means to identify business critical data, so that only the necessary files are committed to long-term storage,. At the same time, the non-critical data must also be tracked, so that machines can be fully recovered in case of machine failure.

In this paper we will discuss the concept behind our system of intelligently managing redundant and unneeded information in the enterprise. Our method employs a rule system that allows us to reduce or eliminate the redundant and useless information that we backup while still providing quick restore times in the event of data loss.

1. Introduction

Experience is showing us today that in the Windows NT class machines, historical backup systems are ill equipped to deal with the massive growth we are seeing in both the sheer number of client workstation machines to manage, as well as the size of the data to be backed up.

As the industry finds larger NT/95/98 client deployments with larger, cheaper, newer technology hard drives, we discover that in large corporate environments, more than half of the information that we commit to backups is considered redundant or short term information. This data does not need to be committed to tape storage since it either exists in other areas of the enterprise, or is used for short term reasons such as Internet cache files or temporary work files.

We have identified several data types that can be handled in a much more efficient manner, through the definition of an intelligent rules system. Rather than backing up all of the data in the enterprise all of the time, we can backup only the data that matters, and reduce the amount of redundant and unneeded files that we are committing to backup via a complete inclusion/exclusion rules system.

This system also provides us with a means of allowing users the ability to define their own exclusions and inclusion in the backup set as well as a level of server dictated excludes and included based on commonality or criticality for information and files.

2. Common File Determination

In a normal enterprise with 5000 machines, each of the machines will contain a local copy of Windows NT/95 System files, and will also probably contain a multitude of workflow applications that the user keeps locally.

All of these system files, workflow applications, and cache files lend to the large amounts of data that is stored locally, and subsequently backed up to ensure that we can recreate a users machine in the event of data loss or total system loss

Many would argue that users could themselves exclude the directories and files that are redundant and do no need to be backed up, but Windows NT and Windows 95/98 lacks the concept of a user directory where user information is stored locally for the users applications and data. Thus applications and users are free to disperse their applications and data all over the structure of their local storage device, and experience has shown that this often occurs.

This makes a user driven visual exclusion system extremely unreliable, and by itself, is clearly insufficient as the sole means of determining file/directory level exclusion. In addition, the user may unknowingly exclude a critical area of their storage device thinking that only application data was stored in this area, while really also excluding from the backup system critical information needed to recreate their computing environment in the event of data loss. Finally, the users frequently fail to utilize the system at all, rendering it useless.

The answer would seem simple. When backing up the client, don't have it send files that are considered redundant, and in the event of data loss or a total system failure the server gets the needed redundant information from another tape that did have this information backed up.

This isn't a foreign concept, and many backup vendors are using this approach such as IBM's ADSM and Seagate Software's Palindrome software's "Tower of Hanoi" system. The basic premise that they work on is keep a list of files that have been backed up *X* number of times and once *X* has been reached, the server then instruct the clients to not send those files during the backup session. However these systems make an assumption that large tape changers or storage facilities are on-line. This is because a file that might be referenced for a restore would only exist on a few backup sets in the enterprise backup system. In the event of a large amount of information to be restored from tape such as an application directory, or a collection of application directories the number of tapes needed for the restore could become very large for a relatively small amount of information to be restored. This is because you could encounter massive data dispersion across the restore set referencing a multitude of tapes to recreate an area or an entire local storage device for a user.

2.1. Finding the Unique Components in the Commonality Set

In our research on how to combat this problem we looked at the file/directories across 3500 Windows NT machines in our local LAN environment that had similar applications installed on them.

The total amount of information scanned was approximately two terabytes of information across 3500 machines. After we indexed the information across all of the machines in the enterprise we found that almost 61% or 1220 gigabytes of information was common to more than one machine. After determining the size of the common information contained in the enterprise, we then re-indexed the common information to find out how much unique information was contained in the 1220 Gigabytes of information.

Example:

> The installed version of Microsoft Office is 300 megabytes of information loaded on a user machine.
>
> ***MSO=300 Megabytes on single machine***
>
> Microsoft Office is then installed on 3500 machines, so the total amount of combined space across the enterprise would be:
>
> ***MSO X 3500 Machines = 150GB***
>
> So the total amount of space to backup Microsoft Office across the enterprise would be 150GB in storage costs, but the unique amount of that common information was only 300MB.

So by applying this method on index of the total information in the enterprise that was common (*1220GB*) we find that by reducing this information down to the unique components in the commonality set, we find that for the 1220GB of common information that only 18GB of that information is unique to the commonality sampling of data.

2.2. Centralized Storage of Redundant Information

By seeing that the amount of unique information that is contained in the redundant information set is approximately two percent of the overall size of the redundant information we find that we could keep this information on-line so that the restore engine could draw from this common file repository for common files instead of having to request multiple tapes from a library system or off-line vault.

Since a dramatically less amount of workstation information will no longer be sent across the network to the backup system, we will have a dramatic reduction in the

impact on the network bandwidth in the LAN as well as quicker backup and restore times since the amount of data being transferred has been drastically reduced..

Tape cost in the enterprise will also be reduced since less information is being sent across the wire to the backup host. This results in less information that will have to be committed to tape since common information will be stored on a network shareable fileserver that backup/restore system will use for referencing common files during restores.

3. "Unique Trash" Identification

During the initial data collection of machines in the enterprise we also isolated another source of potential unneeded data that is being passed from the backup client to the backup host that would not fall within the commonality exclusion engine because of the unique nature of the data.

Many current internet applications and browsers utilized a local client side caching system where they can store commonly used files, images, and meta information that is passed to the client during web browsing or client activity.

Although this cuts down on the use of network bandwidth to the local workstation, this cache information can collect over time, and even when most local client side caching is set to a minimum of 10 megabytes on the client, if we were to multiply that across 5000 machines in the enterprise that would mean that we are backing up on the order of 50GB of information each time we do a full backup of a machine. Do to the randomness of this information caused by each client visiting different Internet sites and looking at different information, their local collection of this information would vary from machine to machine, thus negating our commonality exclusion rules system.

So now we must enact a new exclusion rule to handle this "unique trash" that would not normally be excluded by a commonality engine, and must then be picked up by an application specific unique trash rule.

As an example:

Possible unique trash filters for common applications

ServerExclude
C:\ProgramFiles\Netscape\Cache.**
C:\Temp.tmp*
C:\Temp.~oc*

4. Incorporating the Rules System

The ability to identify the large amount of redundant data does not, by itself, give us the ability to develop a software package that takes advantage of the research that we performed. In order to do anything useful with the information we had gleaned from our network, it was necessary to create a system by which we could define what we wanted included or excluded in out backups. This system had to be both compact and easily interpreted, to minimize both the time needed to load the information and to process it. The system also had to be ultimately flexible, to allow for any and every combination that might occur. Finally, the system had to handle questions of precedence, so that conflicts between different rules (as we chose to call our definitions) could be quickly resolved.

This rule system is designed to provide the ultimate in flexibility in determining which files to back up and which to ignore, without overwhelming the user with complexity. In this system, a rule refers to some definition that describes a set of files. That set may consist of one file, several files, or no files. In addition to defining a set, a rule also describes an action to be taken with that set, either to include or to exclude those files from the backup set.

To provide the extreme level of flexibility that is part of this system, rules are divided into a total of twenty-five categories. With the exception of one rule category, all of these categories can be defined by a set of four characteristics: source, scope, direction, and type.

Source
Refers to whether a rule is defined as a client-level rule, or a server-level rule.

Scope
Determines if the rule applies to an explicitly named file, or a group of files defined using wildcards.

Direction
States if the rule defines an included set of files, or an excluded set of files.

Type
States if the rule applies to the files in a single directory (hereafter known as a file level rule), to all files in a given directory tree (hereafter known as a directory level rule), or to all files on a given machine (hereafter known as a global level rule)

<u>Rule Categories</u>

These four characteristics can be combined into twenty-four different categories of rules, each governing a distinct set of files. The combinations, and what they encompass, are described below:

Server Explicit File Inclusions and Exclusions
A rule in either of these categories will refer to a single fully qualified file name.

Server Wildcard File Inclusions and Exclusions
A rule in either of these categories will refer to a set of files in a given directory.

Client Explicit File Inclusions and Exclusions
A rule in either of these categories will refer to a single fully qualified file name.

Client Wildcard File Inclusions and Exclusions
A rule in either of these categories will refer to a set of files in a given directory.

Server Explicit Directory Inclusions and Exclusions
A rule in either of these categories will refer to all files in a given directory tree.

Server Wildcard Directory Inclusions and Exclusions
A rule in either of these categories will refer to a subset of the files in a directory tree.

Client Explicit Directory Inclusions and Exclusions
A rule in either of these categories will refer to all files in a given directory tree.

Client Wildcard Directory Inclusions and Exclusions
A rule in either of these categories will refer to a subset of the files in a directory tree.

Server Explicit Global Inclusions and Exclusions
A rule in either of these categories will refer to all occurrences of a given file on a system.

Server Wildcard Global Inclusions and Exclusions
A rule in either of these categories will refer to all occurrences of a set of files on a system.

Client Explicit Global Inclusions and Exclusions
A rule in either of these categories will refer to all occurrences of a given file on a system.

Client Wildcard Global Inclusions and Exclusions
A rule in either of these categories will refer to all occurrences of a set of files on a system.

The twenty-fifth rule category is a special case that is always server defined, is always an include, always defines an explicit file name, and always applies to the entire machine. How it differs from the other rules is described below. It is called an "Always Include Rule."

4.1. Rules Priority System

In order for this system to work, a priority system is also defined, giving each rule category a level of importance with regards to other categories. This allows for a means of simply defining very complex backup sets. In determining priorities, the following considerations were used:

1. Given two rules where only the source differs, the server rule has higher priority.
2. Given two rules where only the direction differs, the inclusion rule has higher priority.
3. Given two rules where only the scope differs, the explicit rule has higher priority.
4. For rule types, file rules are higher priority than directory rules. Global rules are handled slightly differently.

Below is a table giving the priority levels and the rule categories at each level:

Level	
5	Server Explicit File, Server Always Include
4	Client Explicit File, Server Wildcard File, Server Explicit Global
3	Client Wildcard File, Server Explicit Directory, Server Wildcard Global
2	Client Explicit Directory, Server Wildcard Directory, Client Explicit Global
1	Client Wildcard Directory, Client Wildcard Global

When defining the priority scheme, file direction is used as the tiebreaker for rules at the same priority level. The twenty-fifth category is given the highest priority, due to its special nature.

It is possible to define a system with a different priority set. It is also not necessary for every rule category to be present or utilized.

The Always Include rule is used by the administrator to define files that must be backed up every time the server connects to the client, regardless of any other considerations, including user level excludes, or full/differential backup type. This is to guarantee that the most recent backup tape will always have this file, making the restoration of that file easier.

5. Optimization

There are two types of backups performed in our system: full and differential. The full backup encompasses the entire contents of the hard disk, taking into account any exclusions that are in force. The differential backups contain all files that are new or modified since the last full backup. This system is designed to minimize the data backed up, while at the same time making it as simple as possible for restores to be performed. It is possible, however, for the following situation to occur: A day or two after a full backup is performed, the user installs a huge number of files on the system. Using the backup scheme defined above, these files will be backed up every day until the next full, which may not occur for another four weeks. To solve this problem, we follow the following procedure: When the server contacts the client to perform a backup, it asks for an estimate of the backup size. The client returns both the full backup size, and the differential backup size. If the size of a differential backup would exceed a user-defined percentage of the full backup size, then the full backup is performed instead of the differential. By defining a maximum time between full backups of thirty days, we also guarantee that the last full is more accessible for restores.

6. Conclusions

Our conclusion at the end of this study was to implement our inclusion/exclusion rules system in our corporate backup system to reduce backup costs in hard dollars and infrastructure.

We are fortunate in the fact that we use an internally developed system for backing up Windows NT/ Windows 95/98 workstations in the enterprise and have complete control over our own software for clients and servers.

These modifications at the time of this writing are being integrated into our existing product for in house use and we expect to see it come to full production use sometime in the summer of 98 in our corporate LAN environment.

7. Acknowledgments

The entire development and design team would like to thank Qualcomm who made this backup project possible, and to our manager TJ Fiske who had confidence in our coding Kung Fu.

Also a special thanks to the engineering teams at Compaq, and JD Marymee at Novell Inc. for his big brain.

Providing Reliable NT Desktop Services by Avoiding NT Server

Thomas A. Limoncelli, Robert Fulmer, Thomas Reingold,
Alex Levine, Ralph Loura*
Lucent Technologies, Bell Labs
Murray Hill, NJ, 07974
ntdesk@research.bell-labs.com

Abstract

We have developed a reliable, stable NT Desktop environment for our customers. The services we provide include: Standard desktop applications (word processing, spreadsheet, etc.), access to UNIX compute servers, file storage and backups, e-mail, printing, calendar, netnews, web, and Internet access. We founded our architecture by selecting open, standard protocols rather than specific applications. This decoupled our client application selection process from our server platform selection process. We could then choose the server based on our needs for reliability, scalability, and manageability and let customers independently choose their clients based on their needs of platform (NT or UNIX), features, and preferences. We can now choose between competing server products rather than be locked into the (potentially difficult to manage) server required for a particular client application. This created a "no compromises" environment on the desktop as well as in our server room. Our customers are happy because the "tail" doesn't "wag the dog". Our ability to manage this infrastructure is superior because the dog doesn't wag the tail either. The resulting system gives us a strong base to build new services.

1 Introduction

Our department is responsible for providing desktop and back-end (server) computer and network services for approximately 600 customers in the "research" part of Bell Labs, a division of Lucent Technologies. We have a very stable and manageable system that can scale as needed. In this paper we hope to refute several myths: (1) It is impossible to provide reliable NT Desktop services. (2) It is impossible to integrate NT and Unix into one coherent environment. (3) Adding NT desktops means getting rid of all UNIX back-end servers. We prove these by demonstration.

*Currently an independent consultant

2 Background

Bell Labs is biased towards open systems and publicly available standards. As the inventors of UNIX, we have a large and stable UNIX environment. When PCs first appeared they were often castigated to separate networks for security reasons and we required users to support themselves. Demand for official support grew at the same time as self administered machines were causing havoc. Finally we decided to give them official support to limit their damage. As NT grew in popularity in the industry, our customers needed to use it and we integrated it into our environment.

Our users (whom we call "customers" [Smallwood]) expect us to provide certain services and leave them alone to be self-sufficient for selecting their own tools, etc. We have a very technical customer base. For example, we provide file service but they select their own development environment. Since they preferred this with their UNIX environment, we adopted the same policy as we began official support of PCs. There are three categories of services we provide to our PC users.

The first category of services is related to deployment. We install new PC hardware, load the operating system, connect it to the network, and install and configure all applications. We also handle all account creation/deletion services and manage the

NT Domains. These are outside of the scope of this paper, but are touched on as appropriate.

The second category is the main desktop applications that we provide. These include office automation, e-mail, calendar, web browser, web publishing tools, and access to UNIX compute servers (via X windows and telnet).

The third category is the "back end" centralized services we provide such as file storage (with access from NT or UNIX), backups and restores, printing, netnews (bulletin board system), web servers (intranet and external), and Internet access.

3 The Philosophy

Two philosophical rules guide our decisions as we develop our environment. The first rule is to select open systems and protocols whenever possible. The other is to keep things simple. We have learned these lessons the hard way, and now they serve us well when we follow them.

3.1 The Rule of Open Systems

Customers want an application that has the features and ease of use that they need. System Administrators (SAs) want an application whose server is easy to manage. Traditionally either the customers or SAs have more "power" and make the decision in private, surprising the other with the decision. If the SAs make the decision the customers consider them fascists. If the customers make the decision it will no doubt be a package that is difficult to administer which will make it difficult to give excellent service to the customers.

There is a better way that strikes a balance that lets everyone win. We select protocols based on open standards and permit each "side" to select their own software. This decouples our client application selection process from our server platform selection process.

Customers are free to choose the software that best fits their own needs, biases, and even platform. We have a corporate standard client (software) that receives official support but many of our customers are happy self-supporting their own rogue selections. We can not force people to use software they don't like, so we must "use the carrot, not the stick" and draw customers to our recommended software with incentives of reliability and support.

We SAs can independently choose the server based on our needs for reliability, scalability, and manageability. We can now choose between competing server products rather than being locked into the (potentially difficult to manage) server software and platform required for a particular client application. In many cases we can even choose our hardware and software independently if a vendor supports multiple hardware platforms.

For comparison, the opposite strategy would be to let the customers select the application without the informed consent of the staff that would be running the servers. For example, a local (New Jersey) pharmaceutical company selected a particular proprietary e-mail package for their PC users after a long evaluation. Their selection was based on user interface and features with no concern for ease of server management, reliability, or scalability. The system turned out to be very unreliable when scaled to a large number of users. In particular, data corruption problems were frequent and result in having to send the e-mail database to the vendor through the Internet for de-mangling. The system also stores all messages from all users in a single large file which must be kept writable by anyone, which is a security nightmare. Because the package is not based on open protocols the system support staff can not seek out a competing vendor which would offer a better, more secure, reliable, server. Because of the lack of competition the vendor considers server manageability low priority and ignores the requests for server-related fixes and improvements.

The answer is to strike a balance by decoupling the client and server selections. Open protocols permit us to do this because we can select clients from one vendor and servers from another. These two vendors' products talk to each other because the protocol between them is created in an open forum and is publicly documented. Anyone could make a compatible client or server. Consumers can choose between any number of clients or servers. End-users can select from an array of clients, possibly even switching between different ones for different tasks. SAs gain similar advantages. Vendors of server software are forced to compete with each other on a level playing field [Fair]. If the current server software begins to lag behind the competition, SAs can opt

to switch to an alternative vendor without forcing users to change clients. Vendors are more responsive to their customers when they know that their customers can leave them without significant pain.

Critics would say that the customers are the center of the universe and therefore their needs override any concerns of the IS staff. The IS staff should be able to learn any system that the customer selects. However, the reliability and scalability of the server is as much an issue to the customer as is a good user interface. Customers may not feel scalability is important, but they will understand that a mail system flooded by chain letters should not buckle and be down for a day as it is repaired. They might not be concerned by whether or not the IS staff will find it easy to manage the server. However, they will be frustrated if they have to wait a week for what seems to be a simple request, but is actually a major undertaking due to the way the server was designed. All of these "secondary" issues are important to the customer but usually only after a disaster has educated them the hard way. The SAs have a responsibility to present these concerns to the end-users in hope that they will be adopted as concerns of their own. To do this the SAs must partner with customers, use the customers language, not technical jargon and other techniques described in [Bashein].

3.2 The Rule of Simplicity

Supporting a mixed NT and UNIX environment is very complicated. Others have met this complexity by building larger, even more complicated systems to address the various issues. We feel that is the wrong direction. We looked to break the problem into smaller, more simple, chunks. Simple chunks can be implemented. Difficult chunks can be thought about, pondered and researched until we find simplifying principles and constructs that turn them into simple chunks. If something can not be simplified we would rather leave it unimplemented than create a monster that can not be tamed.[1]

While this sounds like we leave a lot unimplemented the opposite is true. Delaying the difficult chunks gives us more time to implement the first chunks "the right way". When those are complete, we have a better understanding of the system and those "more difficult chunks" become easier to simplify. Often we discover that those "difficult chunks" were hogwash that were not needed anyway. Either way, our strategy achieves more because we can remain focused on a smaller set of issues at a given time. It is with great hubris that someone thinks they can plan out an entire system without the benefit of the knowledge gained by first having solved smaller portions of the problem.

[1] A cynical version of this is described as "The New Jersey Approach" in [Gabriel].

4 The Services

We will now explain how we engineered each application using those two philosophies. Some fit easily into our philosophies. Others presented challenges.

4.1 E-Mail

The protocols we require for e-mail are the historic Internet standards that the world uses:

> Transport of mail must be via Simple Mail Transport Protocol [RFC821]
>
> Mailboxes must be accessed over the network via Internet Message Access Protocol - Version 4 (IMAP4) [RFC1730]
>
> Mailboxes must be stored in UNIX Mail format and be accessible from a UNIX platform

To meet these requirements, our supported client is Netscape Mail, which access mailboxes via IMAP4, on PC or UNIX. Some UNIX-oriented users use `elm`, `mh`, `mutt`, or even `/usr/ucb/mail`. Our servers are Sun Solaris 2.6 machines running Sendmail 8.8.8 [Allman] as an Mail Transport Agent (MTA) and procmail [Berg] as a Mail Delivery Agent (MDA). IMAP4 protocol is supplied by Sun's SIMS 2.1 IMAP4 Server product. We are evaluating the University of Washington IMAP4 server and may switch to it in the future. Because both IMAP4 servers store mailboxes in UNIX Mail format, we can change servers with little effort.

We selected Solaris over NT because UNIX scales better, is faster, can be made more secure, and it is easier to debug problems [Standish] [Kirch] [Petreley]. Sendmail is one of the few MTAs that we know of that is flexible enough to handle our

complicated configuration requirements. Our alias management system is complicated due to our large size (we import alias information from many sources to build our alias database). We have a fine, robust mail system via our UNIX servers. Why reinvent the wheel when we can give NT users access to our current wheel? Open protocols permit us to do just that.

4.2 File Services

Depending on our customer needs, different file service options are available. Some customers need access only from NT, others only from UNIX, others need to access their files from both. We feel that eventually all customers will need access from both; even UNIX-only users will want to share data with NT-only users.

NT clients access file servers using the Common Internet File System (CIFS) protocol [Leach] (which is Microsoft's new name for the Server Message Block (SMB) file protocol [SMB].) UNIX clients use the Network File System (NFS) [Stern] protocol. Some file servers support one or both of these protocols. File servers come in all sizes from small to extremely large. We feel at this time the file service marketplace can be summarized as in Figure 1.

Large	EMC Auspex		
Medium	NetApp	NetApp	NetApp
Small	Sun	SAMBA	NT Server
	NFS-Only	*Both*	*CIFS-Only*

Figure 1: The File Server Market

For small-scale NT file service, NT Server is appropriate. A UNIX Server is appropriate for small-scale NFS service. If the data must be accessed by both, a UNIX server running SAMBA [SAMBA] or Syntax TotalNET [TAS] is fine. We have multiple terabytes of data and it almost always needs to be accessed from both kinds of clients. Therefore those solutions have been nearly phased out in the past year.

For medium-scale file service with CIFS and NFS we choose Network Appliance Filer (referred to as the NetApp Filer) [Hitzl] dedicated file servers. A typical user has a directory on a NetApp Filer that is exported via NFS for access from UNIX and as a "share" available to NT systems. Some customers have requested to have the share only be a portion of their directory structure, usually a sub-directory called "PC". We get this request less often now. We are very happy with the NetApp Filers' ability to solve the problem of integrating NT and UNIX. On top of that we get snapshots, RAID and other features. Performance is exceptional for NFS as well as CIFS, almost dispelling the general perception that CIFS's design prevents fast implementations from existing.

Management of the NetApps is "free" since they access NIS for UNIX account data and NT Domain for NT account data. By keeping user names in sync, these systems require very little new administration tasks. The NetApp "does the right thing."

While the NetApp Filers are not inexpensive, we find their total cost of ownership is on par with other solutions. We occasionally price out an equivalent PC-based server for comparison and generally find the price per megabyte comparable after including RAID and other features. Such a system would not integrate NFS and CIFS as well, nor would performance be as good. Also, with our UNIX background, we find the Network Appliance File Servers easier to manage.

Our UNIX NIS configuration is automated and includes home spun components that update our NetApps (see [Limoncelli] for complete details). In fact, since the updates are automated as part of our NIS push system, additional NetApps require nearly zero additional work once it is included in our NIS database of NetApps. We encourage each large (200-400) group of customers to procure its own NetApp Filer which we then manage.

Because each small group of customers procures its own NetApp Filer, we have not had to evaluate solutions for large amounts of data. We have a large amount of data, but we have developed ways to manage it efficiently as many medium chunks of data.

4.3 Backups

Backups must be reliable. They must be on standard tapes in a format that can be read even without access to the backup software. To be cost effective a single server must be able to back up many machines and daily human activity must be minimal and simple.

We struggled with home grown solutions on our UNIX systems for years often employing full- or part-time staff just to change tapes. Later we adopted commercial software to gain quickly the new features we needed: Robot controlled tape library/jukebox, better tape handling, ability to back up non-UNIX systems, and enhanced index of tape contents.

We selected BudTool from IntelliGuard [BudTool] but other commercial products would serve our needs. BudTool's differentiators are (1) tapes are in UNIX "dump" format and therefore can be read without BudTool; (2) excellent support for NetApps; (3) use of the Network Data Management Protocol (NDMP) [Stager] open standard for for backup systems to access filesystems, tape drives, etc. over the network. This permits our backup servers to back up file servers to local tape drives or to the tape drives on other machines in the network.

BudTool is as complicated to manage as you make it. We maintain a relatively simple configuration and four huge tape jukeboxes and libraries. We have found their on-site support to be invaluable and worth the expense.

We do not back up our desktop systems. Customers know they are not to store data on them. We help customers conform to this policy by providing sufficient network bandwidth and fast file servers. That is, we make it more appealing to keep data on the server rather than being fascist about enforcing the policy. If a desktop disk dies we can replace it and reload the operating system and applications in about an hour using AutoLoad [Fulmer]. NDMP clients for NT are becoming available now so we are re-evaluating our desktop backup policy.

4.4 Printing

We find that printing is constantly fraught with problems due to printer jams and out of control print jobs that need to be cancelled. Doesn't anyone just print ASCII anymore? Network printing is fraught with badly written protocol stacks on the client and printer end.

In particular, most network printers prefer to have only one machine talking to them at a time (this defies our definition of a "network printer" but this is the reality we have to deal with). NT clients lose a lot of printing features unless they are talking directly to NT servers. Our bias is to spool to some central machine (or its hot standby) so that we have a single point of control when jobs need to be cancelled, etc.

Our solution is as follows: UNIX clients funnel jobs to a UNIX print server using the LPD protocol [RFC1179]. NT clients funnel jobs to an NT print server using the native NT print protocols. The NT print server funnels jobs to the UNIX print server using LPD. The UNIX print server is the only machine that directly talks to printers.

The UNIX print server is a Sun Solaris 2.6 host running LPRng [Powell] which is freely available software. LPRng completely replaces the printing system on Solaris, but is backwards compatible. LPRng's strong point is that it compensates for the broken LPR implementations frequently found in today's network devices (both clients that send jobs to it and the network printers that receive jobs). It converts non-PostScript data into PostScript. It detects and compensates for badly formatted PostScript data. It also does a fine job of accepting from and sending to properly functioning LPD implementations.

Funneling all jobs to a single UNIX server means a single spool to access when bad jobs need to be removed. In fact, LPRng can be configured to permit anyone to kill a job in the queue, something we do since we trust our customers. We have one machine spool the print jobs for each group of printers. This machine is a single point of failure but many of our UNIX compute servers can function as a stand-in for our print spooler when needed. All of our configuration files refer to the spooler by a DNS alias (CNAME) rather than the host's real name. If the spooler dies, a simple change to the DNS will direct all print jobs to a replacement machine.

4.5 Access to UNIX Hosts

Access to our UNIX servers is provided by Exceed [Exceed], an X Windows package for NT. This essentially turns the PC into a fully functioning X Terminal. In fact, we have reduced the number of X Terminals we purchase as a PC can cost about the same, yet can run PC applications locally. In particular, Netscape runs much faster on a PC compared to running it on a UNIX server and displaying it elsewhere via X Windows (whether the final display is on a PC or X Terminal.)

4.6 Internet/Intranet Services

Our standard desktops are loaded with Netscape's web browser. Customers choose their own tools to generate web pages; many prefer to use text editors such as `vi` or `emacs` from the UNIX servers. For customers that need collaborative document features, we use Netscape Enterprise Server [NSES] which is a Java-based application (and therefore runs on NT and all UNIX platforms) that lets users edit pages, lock/unlock them, use revision control, and control who may/may not edit a file. Because open protocols are used, our clients and servers can be of different platforms.

We choose UNIX for our web servers because management and scalability is critical. We securely mirror our web sites using "Stage" [Ches] which is only available on UNIX but is available to the public in source code form. Customers that wish to have external web pages maintain them on an internal server which is mirrored to the outside using "Stage". If our external web server were broken into, we could format the disks, reload the software, and "Stage" would copy the web pages back into place.

Our web proxy/cache is a Netscape Proxy Server [NSPS] running on a Sun Sparc Ultra running Solaris 2.6. While we use a transparent firewall [LMF] (i.e. one that does not require SOCKS or other proxies) users are encouraged to point their web browsers at our web proxy because it caches web pages.

Most of our customers prefer to host their web pages on our server. Customers with special requirements run their own web servers from their desktops. If traffic to their web site becomes considerable, they have the option of moving the data to a central server which has better network connectivity. However many of our customers are developing small, experimental CGI or Java applications and a personal web server suits their needs. In this case, they choose which server they prefer to install and run.

Our Discussion/Bulletin Board ("groupware") service is based on the Network News Transfer Protocol (NNTP) [RFC977], an open standard, so that NT as well as UNIX clients may access it. It is our pre-existing netnews server. Our netnews service is provided using InterNetNews (INN) [Salz] on a Sun Sparc Ultra with Sun On-line Disk Suite and enough disk space to store a month of news (except the "adult" stuff).

4.7 Calendar Management

There is a trend to do more team-based collaborative work in Bell Labs. That means more meetings than ever before. We now facilitate this process with a shared calendar server. Since there are no ratified standards for calendar server/calendar client interaction, we placed our bets on a vendor that we believe will adopt the IETF standards as they are born. We have an additional requirement that the client must be available for Sun Solaris, SGI IRIX and NT. This reduced our choices to Netscape Calendar Server and Client. We run the server on a Sun Solaris 2.6 server.

This is a recent addition to our network and so far we are happy with the fact that NT, Solaris and IRIX users can finally share calendars. Our Palm Pilot users are ecstatic with the interoperability.

4.8 Name Service and Account Administration

Our NT Domain service is provided by a pair of NT servers which are our Primary Domain Controller (PDC) and a Backup Domain Controller (BDC). These machines are also our Windows Internet Name Service (WINS) servers. We have "engineered them for reliability" in stupid, brute force ways because, unlike our UNIX servers, they can not easily be remotely rebooted or maintained. For example, installing new software often requires a reboot. Reliability is achieved by running no other services on them. We are unhappy that reliability has to be achieved this way. We are investigating other options: new software for UNIX servers

that turn them into PDCs; replacing NT Domain all-together with Light-weight Directory Access Protocol (LDAP) [RFC2251], an open standard for directory services; and other options.

Domain Name Service (DNS) [RFC1035] and Dynamic Host Configuration Protocol (DHCP) [RFC2131] services are provided by our existing UNIX servers. We require authenticated logins (single-use passwords via Hand Held Authenticators) to these machines since so much depends on DNS being reliable and authentic. For DHCP we use ISC's free DHCP reference implementation [DHCP] and are extremely happy with its flexible configuration file format. We actually generate the configuration file from our NIS data with a `perl` script. SAs don't actually have to know how to modify the DHCP database. They enter certain information and a perl script generates the rest. We also use the ISC "BIND" DNS software [BIND].

5 No "single log on" yet

As we mentioned earlier, we aim to simplify the complex and delay implementing what we haven't yet simplified. The challenge of a single network login is in that later category.

Currently every user has two accounts, a NT Domain login and a UNIX (NIS) login. While other NT and UNIX integration papers have focused on integrating logins we saw it as a holy grail that would waste our time if we pursued it.

We chose to use human discipline instead. That is, our SAs know to always use the same user name for a customer when creating their NT and UNIX logins. That is, my NT Domain user name is "tal" and my UNIX NIS login is also "tal". If I wish to change my password, I must do it twice, once for each system. (Or, I may choose to maintain separate passwords).

The ability of a NetApp to be programmed so that that "tal" in UNIX is "tal" in NT means the problem of dual platform file access is solved automatically. (NetApp permits us to specify exceptions to this rule if need be.) File permissions are handled like magic based on which protocol the request came from. That is, a request received by CIFS has NT file semantics and a request received by NFS has UNIX (POSIX) semantics [Hitz2].

In hindsight, if an integrated NT and UNIX environment simply means the same user name gets you to the same set of files then we achieved our goal without trying. Thus demonstrating the superiority of our "delay the complex" philosophy. We "missed the bullet" on that one.

If you feel that we are rationalizing the fact that we shirked our responsibility to achieve 100% perfect integration we have two responses:

First, we invite you to interview our customers who feel we have provided for the integration they needed. If their needs are met, the features we didn't complete aren't needed or we can surprise and delight them with such features when they do arrive.

Secondly, we could have spent the last two years developing a single login system and not had time to complete the other services we created. By the time we would have been done, LDAP (an open standard) has arrived. We would be left with a home-grown, incompatible single-login system that would break with every new release of NT and fewer of our other services would be complete. Instead we have a solid foundation to build on and are ready to embrace the coming third party products based on the newly developed standards. We are currently investigating the new single-login options available to us.

6 Protocol Summary

Figure 2 summarizes each application, the protocol selected, and the client and server used. We feel we accomplished our open protocol goal as well as possible given the challenges presented to us and made compromises only when essential. For example, while the CIFS protocol is relatively closed (compared to, for example, NFS) it would not have been cost effective to replace the file service client software on the NT systems. In this case, simplicity suggested that we let clients use the protocol that they use best rather than shoehorn them into a new protocol. The ease of use features of NT's native printing system forced us to use it on clients but only to bring print jobs to a central machine that would store the jobs then forward them using an open protocol. Some protocols were in-house (stage) while others are experimental (the calendar-related protocols). Overall, we were able to build the environment we wanted and do so by using open protocols.

Application	Protocol	Open/Closed	Preferred Client	Server
E-mail Reading	IMAP4	Open	NS Mail	Sun SIMS
E-mail Transport	SMTP	Open	n/a	Sendmail 8.8.8
File Service (UNIX)	NFS	Open	Native UNIX OS	NetApp ONTAP
File Service (NT)	CIFS	Closed	Native NT OS	NetApp ONTAP
File Backups	NDMP	Open	NetApp, UNIX, etc.	BudTool
Printing	LPD / PostScript	Open	LPRng*	LPRng
UNIX Access	X / Telnet	Open	Exceed	n/a
Web Access	HTTP	Open	NS Communicator	LMF Firewall
Web Servers	HTTP	Open	NS Communicator	NSES on UNIX
Web Mirroring	Stage	Free	`stage`	`staged`
Collaborative Publishing	HTTP/FTP/Java	Open	NS Communicator	NSES on UNIX
Bulletin Board	NNTP	Open	many/any	INN
Calendar Mgmt	CAP, CIP, etc.	Open	NS Calendar	NS Cal Server
Host Name Service	DNS, DHCP	Open	n/a	ISC DNS/DHCP
Login/Directory	NT Domain	Closed	n/a	NT

Figure 2: Summary of Protocols/Applications Used

7 Conclusion

We took a solid UNIX environment and layered NT desktop services on top of it and created a rich, tightly integrated environment of NT and UNIX computers. It was critical to base our architecture on open standards for protocols rather than specific applications. We then could choose the server based on our needs of reliability, scalability, and manageability and let customers independently choose their clients based on their needs of platform (NT or UNIX), features and preferences. We selected Sun Solaris (or other UNIX variant) or NetApp for almost all of these back-end platforms. We only resorted to NT Server for PDC's, BDC's and NT Print Services. We provide support for the client applications we have evaluated to be the best business choice and require self-support for all others. This encourages conformity to our recommendations without being fascist. The result is an extremely manageable, reliable and scalable NT desktop service and extremely happy customers.

8 Acknowledgments

System Administration is a collaborative effort. We could not have done this work without the involvement of all of our SA team as well as our SA counterparts at our Holmdel, NJ facilities. The feedback from our users was invaluable. Special thinks to Josh Simon for his excellent editing.

References

[Allman] "Sendmail". Eric Allman. `http://www.sendmail.org/`

[Bashein] *A Credibility Equation for IT Specialists.* Barbara J. Bashein, M. Lynne Markus. Sloan Management Review, 38:4. 1997.

[Berg] `procmail`. S. R. van den Berg. `ftp://ftp.informatik.rwth-aachen.de /pub/packages/procmail`

[BIND] BIND. Internet Software Consortium. `http://www.isc.org/bind.html`

[BudTool] BudTool. IntelliGuard, Inc. `http://www.intelliguard.com/`

[Ches] "Cget, Cput, and Stage". Bill Cheswick. Proceedings of the USENIX 1997 Annual Technical Conference. January, 1997. `http://www.bell-labs.com/topic/swdist/`

[DHCP] DHCP. Internet Software Consortium. `http://www.isc.org/dhcp.html`

[Exceed] Exceed. Hummingbird Communications, Ltd. `http://arctic.www.hummingbird.com /products/exceed/`

[Fair] "Software versus Protocol (or file format) Standards". Erik E. Fair, May 1998. `http://www.clock.org /~fair/opinion/open-standards.html`

[Fulmer] "AutoInstall for NT". Robert Fulmer. Proceedings of the 2nd Usenix Windows NT Symposium. August, 1998.

[Gabriel] *The Rise of Worse Is Better.* Richard Gabriel. reprinted in *The UNIX Haters Handbook* appendix C. IDG. 1994.

[Hitz1] "An NFS File Server Appliance" Whitepaper. Andy Watson. Network Appliance, Inc. `http://www.netapp.com/technology/architecture.html`

[Hitz2] "An Integrated Model for NT and UNIX File Service". Hitz, Allison, Borr, Hawley, Muhlestein. Proceedings of the 2nd Usenix Windows NT Symposium. August, 1998.

[Kirch] "Microsoft Windows NT Server 4.0 versus UNIX". John Kirch. June 1998. `http://www.kirch.net/unix-nt.html`

[Leach] "Common Internet File System (CIFS/1.0) Protocol". Paul J. Leach, Dilip C. Naik. December, 1997. `draft-leach-cifs-v1-spec-01.txt`

[Limoncelli] "Building a network for Bell Labs Research South". Tom Limoncelli, Tom Reingold, Ravi Narayan, and Ralph Loura. Proceedings of the Eleventh Systems Administration Conference (LISA '97). October, 1997.

[LMF] The Lucent Managed Firewall. `http://www.lucent.com/security/`

[NSES] Netscape Enterprise Server. Netscape, Inc. `http://merchant.netscape.com/netstore/servers/enterprise.html`

[NSPS] Netscape Proxy Server. Netscape, Inc. `http://merchant.netscape.com/netstore/servers/proxy.html`

[Petreley] "The new Unix alters NT's orbit". NC World. April 1998. `http://www.ncworldmag.com/ncworld/ncw-04-1998/ncw-04-nextten.html`

[Powell] "LPRng - An Enhanced Printer Spooler System". Patrick Powell, Justin Mason. Proceedings of the Ninth System Administration Conference (LISA '95). September, 1995. `ftp://iona.ie/pub/plp/LPRng/`

[RFC821] "RFC821: Simple Mail Transfer Protocol". J. Postel. August, 1982.

[RFC977] "RFC977: Network News Transfer Protocol". B. Kantor, P. Lapsley. February, 1986.

[RFC1035] "RFC1035: Domain names - implementation and specification". P. V. Mockapetris. November, 1987.

[RFC1179] "RFC 1179: Line printer daemon protocol". L. McLaughlin. August, 1990.

[RFC1730] "RFC1730: Internet Message Access Protocol - Version 4". M. Crispin. December 1994.

[RFC2131] "RFC2131: Dynamic Host Configuration Protocol". R. Droms. March 1997.

[RFC2251] "RFC2251: Lightweight Directory Access Protocol (v3)". M. Wahl, T. Howes, S. Kille. December 1997.

[Salz] "InterNetNews: Usenet Transport for Internet Sites". Rich Salz. USENIX Conference Proceedings. Summer 1992. `http://www.isc.org/inn.html`

[SAMBA] "The SAMBA FAQ". The SAMBA Team. June 1998. `http://samba.anu.edu.au/samba/`

[Smallwood] "Whither The Customer?". Kevin C. Smallwood. ;login: and counterpoint by Rob Kolstad.

[SMB] "CIFS in a Nutshell". Microsoft Corp. January, 1997. `http://www.microsoft.com/workshop/prog/cifs/nutshell.htm`

[Stager] "Network Data Management Protocol". R. Stager, PDC and D. Hitz, Network Appliance. August 1997. `draft-stager-pdc-netapp-backup-04.txt`

[Standish] "SUN Also Rises: Solaris vs. NT". The Standish Group. May 1998. `http://www.standishgroup.com/syst.html`

[Stern] "NFS and NIS". Hal Stern. O'Reilly and Associates. July 1991.

[TAS] TotalNET Advanced Server v5.2. Syntax, Inc. `http://www.syntax.com/totalnet/tasbody2.htm`

NT 3.5 / 4.0 Domains for UNIX

Luke Kenneth Casson Leighton
lkcl@switchboard.net

Abstract

NT domain logins, and some experimental administrative capabilities, have been added to a development branch of SAMBA, the publicly available file/print share program that makes UNIX servers look like Microsoft windows NT server.

Further work is needed, but the goal is to make UNIX look like windows NT, over a network. This will include full UNIX command-line administrative capability as well.

The implications of this are that UNIX will be fully adminsterable by the standard NT server tools (e.g "user manager for domains"; "server manager for domains"), and both UNIX and NT will be fully administerable using HTML (cgi-bin wrappers around the smbclient program).

Some of this functionality (both client and server) is already available. The latest version can be obtained by following the instructions in http://samba.anu.edu/au/cvs.html.

At present, SAMBA and smbclient can only provide or obtain information using DCE/RPC: no capability has been added to administer domain servers. This can (should) only be possible to do by administrators. Adding or changing SAM user accounts or domain groups is encrypted. The "backup domain controller" and "inter-domain trust relationships" also needs to be researched.

Final point: anyone running windows NT who allows SMB access through their firewall (ports 137-139) is strongly advised to look up and enable the "RestrictAnonymous" registry key in the microsoft KB articles, and to look for information on the "red button" bug in NT.

1. DCE/RPC in NT (not Win95)

NT runs an implementation of DCE/RPC over an SMB inter-process communication pipe. It uses an IDL to describe the data structures of the various remote procedure calls. NT opens named pipes such as \PIPE\NETLOGON and \PIPE\ntlsa (to do NT domain logins); \PIPE\samr (to do SAM database replication and administration - e.g using SRVMGR.EXE and USRMGR.EXE); \PIPE\srvsvc (to check on the status of files / connections / shares, and to disconnect users or close files).

Basically, all of the administration tools that can select "domains" or "workstations" in NT server all use DCE/RPC.

Windows 95, and WfWg, do _not_ use DCE/RPC. Where similar functionality can be found, there are equivalent calls made to SMB servers on the SMBTrans2 pipe. Windows NT does not use the majority of these functions. Only occasionally, if a DCE/RPC call is not implemented, will an NT server or NT workstation "thunk" down to using SMB Trans2 calls.

The only attempts seen by Windows 95 to use DCE/RPC is simply to open \PIPE\srvsvc (or \PIPE\wkssvc) and then close it. This would appear to be a method to check whether a machine is, to all network intents and purposes, an NT server or NT workstation (from an SMB file access point of view, a Windows SMB client reacts differently depending on what it determines the server to be. The "determination" step is extraordinarily convoluted, and is a really good example of bad object orientated programming).

The DCE/RPC over-the-wire data is relatively simple to decode. Unfortunately, no-one except Microsoft knows what's being used: you can only infer the meaning of this data from clicking on various admin tools, and seeing what happens: classic network reverse-engineering.

The work in progress is a double-edged sword for Microsoft. Their proprietary system is being understood and implemented in a popular SMB client / server suite for UNIX (SAMBA). One of the medium to long term benefits to Microsoft is that bugs will (and are) being found in their code and their protocols. This can

only improve the reliability and security of a quite impressive implementation of remote server administration.

That this is the most widely used remote administration suite in the world (USRMGR.EXE and SRVMGR.EXE, amongst other tools) makes it all the more important that it be secure and reliable.

2. NT Domain Logins

These are documented in http://www.cb1.com/~lkcl/ntdom/cifsntdomain.txt, and implemented in a pre-alpha version of SAMBA. SAMBA is available under the GPL (Gnu Public License).

NT domain logons are what happens on NT when you press ctrl-alt-delete, type in a username and password, select an NT-domain, and press return. What happens under Win95 is completely different (it uses the SMBtrans2 NetUserGetInfo call, but has the same end result in a completely insecure manner).

The sequence of events is as follows:

- Open \PIPE\ntlsa
- Issue a LsaQueryInfoPolicy, level 3
- Issue a LsaQueryInfoPolicy, level 5
- Close \PIPE\ntlsa
- Open \PIPE\NETLOGON
- Issue a LsaReqChallenge
- Issue a LsaAuth2
- Issue a LsaSamLogon, at the "interactive" level
- Leave the connection open until the user logs off
- Issue a LsaSamLogoff, or close the \PIPE\NETLOGON

The LsaQueryInfoPolicy calls determine whether the server is a member of a workgroup (negative answer to both calls), a member of a domain (positive answer only to the level 3 query), or a primary domain controller (positive answer to both calls). Only if the workstation determines that it is talking to a DC does it proceed further.

A session key is created from the server's response to the LsaReqChal and the client's LsaAuth2 Query, on _both_ the client and server. Each and every subsequent transaction is "signed" with a credential chain generated from this session key. [Unfortunately, the signature is not generated from the contents of the transaction itself, making it easy for someone to compromise Domain logons].

The response from the Domain Controller to the LsaSamLogon Query contains all the information that an administrator creates for a user with USRMGR.EXE for example, the user's profile location.

3. NT and UNIX Administration

The DCE/RPC commands, once understood and implemented, can be used to administer or be administered by, anything that also uses those commands. To the best of my knowledge, currently, only NT and its various ports to UNIX (by AT&T and SCO) use these commands. Oh, and SAMBA.

Stub functionality has been implemented in the SAMBA server code to enable "user manager for domains" (USRMGR.EXE) to view UNIX user accounts, and "server manager for domains" (SRVMGR.EXE) to view files, shares and sessions on a UNIX server.

The smbclient program has been updated to allow it to send DCE/RPC commands. It can be used, usually only with administrator priveleges, to provide exactly the same information as USRMGR.EXE and SRVMGR.EXE. This includes viewing files, shares, sessions, domain users, domain groups and domain aliases. Future versions will provide exactly the same administrative capabilities as these two programs, and more.

By ensuring that smbclient's output is in a machine-readable format, parsing scripts running from cgi-bins can be written that will allow NT (and UNIX servers running SAMBA) to be administered by HTML (WWW) clients.

[small side-note: the original purpose behind writing smbclient was as a "boot-strap", or "testing" tool for smbd, and it is still used for this purpose...]

4. NT to UNIX Mapping

This area is probably going to cause the most pain to administrators of mixed UNIX / NT systems, even though it's a pain at the moment. There are two areas of contention: users, and workstations. Through the use of SIDs NT supports, across all installations, unique (world-wide) user, group and workstation identification. (Adminstrators of NIS+ will be aware

that workstations are also similarly uniquely indentified).

The identification is subdivided into SIDs (security ids) for a domain, and RIDs (relative ids - relative to the SID, that is). A RID can be either a local user or a local group. Regardless, it must be unique within the domain (relative to the SID).

NT also has some common (well-known) RIDs and SIDs that have specific meanings (0x1f4 for the domain administrator's RID; S-1-1 for the World SID: see winnt.h or cifsntdomain.txt for some details).

The various UNIX flavours use 0x0 for the uid of the root superuser. They do not support the concept of a well-known guest account, like NT does: they certainly don't support, as standard, the concept of a "SID".

Probably the easiest way to deal with this is to first convert to using NIS+ (or an equivalent). The reason for this is that both NIS+ and NT support, in some form, the concept of "workstations" as individual users. Each domain must have a SID associated with it, and each user (including workstations) must have a user id. There are two other types of accounts, which are used for inter-domain trust relationships, and for primary / backup domain controller relationships.

SIDs and the well-known RIDs will need to be added to UNIX, somehow, in order to support NT domains. It is envisaged that SAMBA will provide this mapping, in such a way that the UNIX OS need know absolutely nothing about NT domains, and NT need know nothing about UNIX.

It actually doesn't matter what the scheme is, as long as it exists, whereby a set of UNIX accounts, workstations and UNIX groups are uniquely mapped into a SID/RID pair, making them world-wide unique.

Of even more contention at present is the issue of mapping existing NT accounts into UNIX ones. This would be best resolved by having a separate SID for the UNIX domain, and setting up an inter-domain trust relationship with the NT server.

Another scheme is to dedicate an entire UNIX server to be a SAMBA DC. Under these circumstances, ordinary UNIX access would be completely denied, and UNIX uids could then be allocated arbitrarily. This would be ideal for migrating from NT to UNIX.

SAMBA supports NT encrypted passwords through a password database API (see http://samba.anu.edu.au/listproc/samba-technical, thread named "password API needed"). If the NT user gives a correct NT password (fully documented in cifs6.txt), SAMBA allows the user access. The UNIX password is not involved in this process, and the UNIX password database still has to be maintained if the users are to be allowed access to the standard UNIX resources.

An alternative scheme to resolve the UNIX / NT username issue is to have a unique mapping for domain RIDs for users within a UNIX domain, but to have a non-monotonic mapping between those RID and the UNIX uids. For example:

NT user	**NT RID**	**UNIX user**	**uid**
Administrator	0x1f4	root	0
root	0x1000	root	0
sales_usr1	0x1001	salsusr	521
sales_usr2	0x1002	salesusr	521
foouser	0x1003	foouser	522
faruser	0x1004	faruser	523

Each NT username / NT RID is unique, yet the sales_usr 1 and 2 map to the same UNIX user and uid. likewise with root and administrator. This would be implemented by having a database which is maintained in parallel with the UNIX password file, which provides you with the mapping between UNIX uids and NT RIDs.

Identical consideration must be given for UNIX to NT group mapping, bearing in mind also, the fact that NT groups are like users: they can own resources such as files and directories.

5. Short-Term Plans

To research the "user manager for domains" functionality, finishing off the "read-only" side. this will allow viewing of UNIX users and groups with

USRMGR.EXE or smbclient. At present, the user profile information is available through smbd, but the domain group and domain alias information is either stub-code or incomplete.

To research the "server manager for domains" functionality, again finishing off the "read-only" side. This will allow SRVMGR.EXE or smbclient to view open files, sessions, shares and users on the server. The DCE/RPC side of this has been implemented in SAMBA. smbd currently only provides stub information; smbclient fully implements the client side. There are two buttons missing: "replication" and "alerts". These are on two separate DCE/RPC pipes, which I have not yet examined, and intend to.

To publicly encourage the discussion and resolution of the UNIX <-> NT SID and RID issue, and to implement an example mapping in SAMBA [see http://samba.anu.edu.au/listproc/samba-technical].

To rework the DCE/RPC code currently written so that it can be used in a general way, not just in a SAMBA-specific way (use of higher order - sometimes known as callback – functions where appropriate, for some of the enumeration containers. e.g. the "NetrFileEnum" and "SamrQueryDisplayInfo" functions).

To write a PAM (plug-in authentication module) which will allow users of Solaris 2.5 and Linux (the only systems that support PAMs, at the moment, to the best of my knowledge) to log in to (and out of!) NT Domains [see http://www.cb1.com/~lkcl/pam_ntdom].

6. Long-Term Plans

To document the full set of DCE/RPC administrative capabilities currently available in NT server, and to see them implemented in SAMBA (client and server) as a means to test and prove their usability and worth (making UNIX, from a network point of view, exactly like Windows NT 3.51 / 4.0 server).

This will include:

- Domain "primary / backup" relationships and inter-domain relationships
- Adding / creating accounts (first implementation to be SAM accounts)
- Closing of files / disconnecting of users / adding or removing shares.
- Changing SAM passwords from NT workstation.

Once this has been achieved, to then suggest the possibility of extending these calls for UNIX-specific (or other o/s, e.g MacOS) needs, as has been done with the CIFS file access protocol (cifs-unix by SCO - www.sco.com; mac extensions by Thursby - www.thursby.com).

Because the work already done by Microsoft in this field is very comprehensive, and appears to include some long-term planning and some degree of protocol independence, it is not expected that there be any re-design of, or significant additions to the DCE/RPC pipes protocols, for use in UNIX <-> UNIX administration.

7. Strategic Aim

To ensure that the administration and operation of "domain" systems (SAMBA, NT 4.0, NT 5.0, etc.) are reliable and secure. This is to be achieved by ensuring that all critical protocols are fully documented, and available for public review.

NOTES

NOTES

THE USENIX ASSOCIATION

Since 1975, the USENIX Association has brought together the community of developers, programmers, system administrators, and architects working on the cutting edge of the computing world. USENIX conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on new developments in all aspects of advanced computing systems. USENIX and its members are dedicated to:

- problem-solving with a practical bias
- fostering innovation and research that works
- communicating rapidly the results of both research and innovation
- providing a neutral forum for the exercise of critical thought and the airing of technical issues

SAGE, the System Administrators Guild

The System Administrators Guild, a Special Technical Group within the USENIX Association, is dedicated to the recognition and advancement of system administration as a profession. To join SAGE, you must be a member of USENIX.

Member Benefits:

- Free subscription to *;login:*, the Association's magazine, published 6-8 times a year, featuring technical articles, system administration tips and techniques, practical columns on Perl, Java and C++, book and software reviews, summaries of sessions at USENIX conferences, Snitch Reports from the USENIX representative and others on various ANSI, IEEE, and ISO standards efforts.
- Access to papers from the USENIX Conferences and Symposia, starting with 1993, via the USENIX Online Library on the World Wide Web.
- Discounts on registration fees for the annual, multi-topic technical conference, the System Administration Conference (LISA), and the various single-topic symposia addressing topics such as object-oriented technologies, security, operating systems, electronic commerce, and NT - as many as ten technical meetings every year.
- Discounts on the purchase of proceedings and CD-ROMs from USENIX conferences and symposia and other technical publications.
- Discount on BSDI, Inc. products.
- Discount on all publications and software from Prime Time Freeware.
- 20% discount on all titles from O'Reilly & Associates.
- Savings (10-20%) on selected titles from McGraw-Hill, The MIT Press, Morgan Kaufmann Publishers, Sage Science Press, and John Wiley & Sons.
- Special subscription rate for *The Linux Journal* and *The Perl Journal.*
- The right to vote on matters affecting the Association, its bylaws, election of its directors and officers.

Supporting Members of the USENIX Association:

ANDATACO
Apunix Computer Services
Auspex Systems, Inc.
Cirrus Technologies
CyberSource Corporation
Digital Equipment Corporation
Earthlink Network, Inc.
Hewlett-Packard India Software Operations
Internet Security Systems, Inc.
Invincible Technologies Corporation
Lucent Technologies, Bell Labs
Motorola Global Software
Nimrod AS
O'Reilly & Associates
Performance Computing Magazine
Sun Microsystems, Inc.
UUNET Technologies, Inc.
WITSEC, Inc.

Sage Supporting Members:

Atlantic Systems Group
Collective Technologies
D.E. Shaw & Co.
Digital Equipment Corporation
ESM Services, Inc.
Global Networking and Computing, Inc.
Great Circle Associates
O'Reilly & Associates
Remedy Corporation
Sysadmin Magazine
Texas Instruments, Inc.
TransQuest Technologies, Inc.
UNIX Guru Universe (UGU)

For further information about membership, conferences or publications, contact: USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710 USA. Phone: 510-528-8649. Fax: 510-548-5738. Email: *office@usenix.org.* URL: *http://www.usenix.org.*

ISBN 1-880446-96-0